AppBuilder
By Magic Software Enterprises

Magic Software AppBuilder

Version 3.2

System Components Reference Guide

Corporate Headquarters:

Magic Software Enterprises
5 Haplada Street,
Or Yehuda 60218, Israel
Tel +972 3 5389213
Fax +972 3 5389333

System Components Reference Guide

Introduction to System Components

A system component is a programming module written in a native programming language, such as Java C, or COBOL, rather than in the Rules Language. AppBuilder system components are a set of internal components tightly integrated into the AppBuilder C, C#, COBOL or Java runtime environments to accomplish tasks such as manipulating the states of window controls. A rule may use just one component or call several in sequence to perform more complex tasks. The development environment includes several system components to help speed application development. This guide describes the system components.
The system components are organized into these categories:

- **Interface** – These components provide a variety of responses to the end user, from creating a pop-up window to creating a warning message, from changing the color of a field to changing the text of a menu.
- **DDE** (Dynamic Data Exchange) – These components implement a message protocol between two processes; for example, implement a message protocol to dynamically update information in an application based on changes that another application makes.
- **Batch** – These components manage storage and handle input and output of records to access files in mainframe batch applications.
- **IMS** – These components relate to applications that run in an IMS environment.
- **BLOB** (Binary large object) – These components manipulate large-object files, which can be either text large-object files or binary large-object files on the mainframe host.

Refer to Smooth Scrolling with Components for examples on how to use several system components together. For information on writing your own user components, refer to the *Developing Applications Guide* .

Interface System Components

Interface system components provide a variety of responses to the end user through the interface. They perform functions such as creating windows for error and warning messages, changing the color and visibility of push buttons and fields, and properly closing down applications in the case of a system failure. This topic includes:

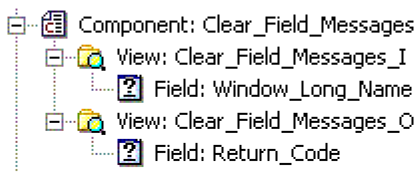
- Hierarchy of an Interface System Component
- Interface Component by Environment
- Interface Components
- Smooth Scrolling with Components

The examples for each component are snapshots of the relevant portion of the calling rule. To make a component functional, the component must be attached to the appropriate rule in the hierarchy and the calling rule must be coded, using MAP statements as required, to manipulate the input and output views of the component. In other words, a component does not stand alone, but must be considered in context of the application and the calling rule.

Hierarchy of an Interface System Components

An interface system component can be seen in the overall application hierarchy in the Hierarchy window of the Construction Workbench. Such a component typically contains one input view and one output view. Each view contains one or more fields. See Component hierarchy for an example of how a system component appears in the Hierarchy window.

Component hierarchy



Component: Clear_Field_Messages
View: Clear_Field_Messages_I
Field: Window_Long_Name
View: Clear_Field_Messages_O
Field: Return_Code

Input View

The input view provides input data to the component. In addition to other fields, an input view may own a field named either Window_Long_Name or View_Long_Name. This field allows you to specify the target window or view on which to perform the action. For example, the component shown in Component hierarchy returns all fields in a specified panel to their non-error condition. To specify the window on which to reset these fields, type the following code into the calling rule:

```
map 'name_of_window_to_reset_fields' to WINDOW_LONG_NAME*
        of CLEAR_FIELD_MESSAGES_I
use component Clear_Field_Messages
```

Some components, such as **Get_User_Workstation_ID** , do not have an input view. These components do not involve a particular window or view, and therefore do not require an input view.

Output View

After you use a component, data resulting from the component's execution is stored in the output view. The data is typically contained in the field Return_Code and other fields. If the component contains sets and values, the value of the Return_Code determines which values (or messages) are displayed in the panel.

Interface Component by Enviornment

The following environments support different components:

- Java Thick Client
- Java Thin Client
- C Language
- C#
- 3270

See also Deferred Interface System Components for a list of the deferred components.
Some system components are not supported in Java client and thin client environments because similar functionality does not exist on the Java platform (for example, DDE components). Other components are not supported due to architectural changes necessary to implement them (for example, Clear_Window_Changes). Note that a subset of the components supported in the C environment are deferred.
Refer to System Components Support Matrix for a complete matrix of supported system components.

3270 Converse Supported Interface Components

The functionality of system components is usually the same, within technological limitations, whether you use them in a CICS or IMS online application (3270 Converse) or a workstation online application. Descriptions of 3270-supported components explain any differences unique to the 3270 execution platform; some components behave differently executing on the mainframe.
All the 3270 Converse Components are linked into the load module HPECO10. The object names are listed in 3270 Converse Components - Object Cross-Reference. The only exception to this is the CGETENV, which is a load module itself. Any system error messages that appear list the object name of the component if there is an error. Use the following table to identify the component to which the error message relates.

3270 Converse Components - Object Cross-Reference

Component Name	System ID	Object Name
Clear_Field_Messages	CCLRFLD	HPECO20
Clear_Window_Changes	CCLRPNL	HPECO21
Get_Altered_Field	CGETALT	HPECO27
Get_Current_Date_Time	CCURTME	HPECO23
Get_Elevator_Position	CGETELV	HPECO28

Get_First_Visible_Occurrence	CGETFST	HPECO30
Get_Full_User_Identity	CGETUIF	HPECO32
Get_Last_Visible_Occurrence	CGETLST	HPECO31
Get_Listbox_Window_Sizes	CLSTWSZ	HPECO34
Get_Menu_Mode_By_ID	CGETMMI	HPECO44
Get_Selected_Field	CGETFLD	HPECO29
Get_Text_Message	CGETMSG	HPECO43
Get_User_Workstation_ID	CGETUWI	HPECO32
HPS_Get_Environment	CGETENV	CGETENV
Set_Control_Color_By_ID	CCTLCID	HPECO24
Set_Control_Mode_By_ID	CCTLMID	HPECO25
Set_Cursor_Field	CCURPOS	HPECO22
Set_Field_Blink_By_ID	CFLDBLK	HPECO49
Set_Field_Color	CFLDCOL	HPECO24
Set_Field_Message	CLLDMSG	HPECO26
Set_Field_Mode	CFLDMOD	HPECO25
Set_Field_Numeric_By_ID	CFLDNUM	HPECO48
Set_Field_Picture	CPICSIZ	HPECO36
Set_First_Visible_Occurrence	CSETFST	HPECO40
Set_Last_Visible_Occurrence	CSETLST	HPECO41
Set_Menu_Mode	CMNUMOD	HPECO35
Set_Menu_Mode_By_ID	CMNMID	HPECO35SET
Set_PopUp_Position	CPNLPOS	HPECO38
Set_Push_Mode	CPSHMOD	HPECO39
Set_Virtual_Listbox_Size	CLSTSIZ	HPECO33
Set_Window_Message	CPNLMSG	HPECO37
Sound_3270_Alarm	CSNDALM	HPECO60

While the smooth scrolling components HPS_Tbl_Init_Size and HPS_Tbl_Get_Data_State are not supported in 3270 Converse, the Set_Virtual_Listbox_Size, etc. may be used to accomplish the same function. |

Deferred Interface System Components

For C applications, some components do not execute immediately when a USE COMPONENT statement executes. These components are called deferred, or buffered, components. Their execution is deferred until the next CONVERSE WINDOW statement executes. These components are deferred only in C applications. Components in a Java environment are not deferred.

Any deferred component executes on the window that is the object of the CONVERSE statement, which may not be the same window as was displayed when the component was originally executed. For this reason, in most cases the fields of the input view cannot be validated when a USE COMPONENT statement executes. Therefore, the Return_Code field of the output view is always set to one (1) indicating success, unless a system-level error occurs, in which case it is set to zero (0). When the CONVERSE WINDOW statement executes, the contents of the input view are checked. To display a warning message in case any abnormality is detected at this time, set the MESSAGES setting in the [AE Runtime] section of the Hps.ini file to "Warning" (case-sensitive) to enable warning messages. For more information about the ini settings, refer to the *INI Settings Reference Guide* .

If an error is detected at this time, a message box indicating the error is displayed.

Several deferred components also have a Window_Long_Name field. If the name mapped to this field is not the name of the next window conversed, the deferred component does not work.

Refer to System Components Support Matrix for a complete list of system components that are deferred in the C execution environment.

C# does not support deferred components.

Java and C# Support for Deferred Components

While some of these components are supported for Java and C# applications, these components do not execute in the same way as they do for C applications. In Java and C#, these components act on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java and C#.

Interface Components

Component Name and System ID table lists the interface system components and the corresponding System IDs:

Component Name and System ID table

Component Name	System ID
Cache_Object	CCACHE
Clear_Field_Messages	CCLRFLD
Clear_Selected_Fields	CCLRSLT
Clear_Window_Changes	CCLRPNL
Get_Altered_Field	CGETALT
Get_Business_Process_Name	CGETBPN
Get_Current_Date_Time	CCURTME
Get_Elevator_Position	CGETELV
Get_First_Visible_Occurrence	CGETFST
Get_Full_User_Identity	CGETUIF
Get_Last_Visible_Occurrence	CGETLST
Get_Listbox_Window_Sizes	CLSTXSZ
Get_Menu_Mode_By_ID	CGGETMNI
Get_Selected_Field	CGETFLD
Get_Text_Message	CGETMSG
Get_User_Workstation_ID	CGETUWI
HPS_Event_Post_to_Child	CPOSTC
HPS_Event_Post_to_Parent	CPOSTP
HPS_Get_Activate	CGETACT
HPS_Get_Environment	CGETENV
HPS_Get_MinMax	CGETMMX
HPS_Get_Window_Handle	CWINHD
HPS_Intercept_Events	CINTEVT
HPS_Limit_Scroll_Size	CLMTSCR
HPS_Set_Activate	CSETACT
HPS_Set_Bitmap_File	CSETBMP
HPS_Set_Cursor_By_ID	CCRSRID

HPS_Set_Help_Topic	CSETHLT
HPS_Set_HTML_File	CSETHTM
HPS_Set_HTML_Fragment	CFRGHTM
HPS_Set_MinMax	CSETMMX
HPS_Set_Selected_Field	CSETFLD
HPS_Set_Window_Attributes	CWINATR
HPS_Tbl_Get_Data_State	CTBLDAT
HPS_Tbl_Init_Size	CTBLSIZ
Restore_Altered_Fields	CCLRALT
Set_CloseDown_Notification	CAPPOFF
Set_Control_Color_By_ID	CCTLCID
Set_Control_Mode_By_ID	CCTLMID
Set_Control_RGBColor_By_ID	CSCRGBC
Set_Cursor_Field	CCURPOS
Set_Default_Push_Button	CDEFBUT
Set_Field_Blink_By_ID	CFLDBLK
Set_Field_Color	CFLDCOL
Set_Field_Message	CFLDMSG
Set_Field_Mode	CFLDMOD
Set_Field_Numeric_By_ID	CFLDNUM
Set_Field_Picture	CPICSIZ
Set_First_Visible_Occurrence	CSETFST
Set_Help_File_Name	CSETHLP
Set_Item_Text	CITMTXT
Set_Last_Visible_Occurrence	CSETLST
Set_Menu_Mode	CMNUMOD
Set_Menu_Mode_By_ID	CMNUMID
Set_PopUp_Position	CPNLPOS
Set_Push_Color	CPSHCOL
Set_Push_Mode	CPSHMOD
Set_Virtual_Listbox_Size	CLSTSIZ
Set_Window_Message	CPNLMSG
Set_Window_Position	CBCKPOS
Set_Window_Timeout	CPNLTME
Set_Window_Title	CSETTIT
Set_Window_Title_Ex	CSETITX
Show_Help_Topic	CSHOHLP
Show_Window_Message	CIMMERR
Sound_3270_Alarm	CSNDALM

Get_Altered_Field

This section presents the purpose, the syntax, the usage, and how the Get_Altered_Field component is supported. See Example: Get_Altered_Field for an usage example.

Purpose

Use this component to determine the screen fields the end user changed. You can use this component iteratively to find all the altered screen fields or call it once to check if a specific field has been altered. If no view name and no field name are specified in the input view, it looks for all changed fields in every view. If a field name is specified but a view name is not, the component searches for the specified field in any view. If a view name is specified but a field name is not, the component searches for any changed field within the specified view.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe but it behaves differently executing on the mainframe than it does on the workstation. A 3270 terminal processes full screens of data, while a workstation processes on a field-by-field basis. Whenever a 3270 screen is redisplayed, all the fields are reset to an unmodified state. Therefore, if this component is not called within a loop before redisplaying the window, the notification about subsequent changed fields is lost. Refer to 3270 Converse Supported Interface Components for a list of supported components.

Syntax

Here is the syntax of Get_Altered_Field component.
Get_Altered_Field

Name	`Get_Altered_Field`
System Identifier	`CGETALT`
Input View Name	`Get_Altered_Field_I`
Input View Field	`View_Long_Name character(30)` The view that includes the field mapped to *Field_Long_Name*
Input View Field	`Field_Long_Name character(30)` The name of the field in the hierarchy diagram to which the field in the window is linked
Input View Field	`Field_Occur integer(15)` Occurrence number of the altered field in an occurring view such as a list box. This field is attached to the system set SEARCH_CRITERIA (SSEARCH), which has the following values: • -1 = Return first altered field • 0 = Return next altered field • *n* = Return only an altered field that is the *n* th occurrence in a list box
Output View Name	`Get_Altered_Field_O`
Output View Field	`View_Long_Name character(30)`
Output View Field	`Field_Long_Name character(30)`
Output View Field	`Field_Occur integer(15)` This parameter directs the component to return the first changed field found, the next changed field, or a field with a particular occurrence number in an occurring view such as a list box.
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Usage

The component returns the view name, field name, and field occurrence of the changed screen field to the calling rule when the component is successful. When no more altered fields are found, the Return_Code field is set to failure. The scope of the altered field search includes edit fields, combo boxes, multicolumn list boxes, check boxes, and radio buttons.
An iterative call to this component loops through all altered fields unless you clear the altered field flags by using Restore_Altered_Fields.

Example: Get_Altered_Field

This example specifies SELECT_CUSTOMER as the view on which to search for altered fields.

```
map 'SELECT_CUSTOMER' to VIEW_LONG_NAME of GET_ALTERED_FIELD_I
use component GET_ALTERED_FIELD
```

Get_Business_Process_Name

This section presents the purpose, the syntax, the usage, and how the Get_Business_Process_Name component is supported. See Example: Get_Business_Process_Name for an usage example.

Purpose

Use this component to return the name of the currently active process.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Get_Business_Process_Name component.

Get_Business_Process_Name component syntax

Name	`Get_Business_Process_Name`
System Identifier	`CGETBPN`
Input View Name	There are no user-defined fields for the input view. The component automatically investigates the present process to determine its name.
Output View Name	`Get_Business_Process_Name_O`
Output View Field	`Business_Process character(8)` For C, this contains the process name specified on the command line when starting the application. For C# and Java, this contains the system id of the root rule. Example: `master -p {process name} -r {rule id}` where `rule id` is the rule implementation name, which is normally its system id unless changed.

Example: Get_Business_Process_Name

This example maps the current process name to the variable THIS_PROCESS.

```
map BUSINESS_PROCESS of GET_BUSINESS_PROCESS_NAME_O to THIS_PROCESS
use component GET_BUSINESS_PROCESS_NAME
```

Get_Current_Date_Time

This section presents the purpose, the syntax, the usage, and how the Get_Current_Date_Time component is supported. See Examples: Get_Current_Date_Time for an usage example.

Purpose

Use this component to obtain the current date and time in both integer format and character format.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Get_Current_Date_Time component.

Get_Current_Date_Time component syntax

Name	`Get_Current_Date_Time`
System Identifier	`CCURTME`
Input View Name	This component has no input view.
Output View Name	`Get_Current_Date_Time_O`
Output View Field	`Current_Time integer(31)` Returned in the format *+0000hhmmss* where: • *hh* is hours • *mm* is minutes • *ss* is seconds
Output View Field	`Current_Date integer(31)` Returned in the format *+00ccyymmdd* where: • *cc* is century • *yy* is year • *mm* is months • *dd* is days
Output View Field	`Curr_Dte_Tm_String character(24)` Returned in the format *day mon dd hh:mm:ss ccyy* where: • *day* is day of week (Sun, Mon, Tue, Wed, Thu, Fri, Sat) • *mon* is month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec) • *dd* is days (00--31) • *hh* is hours (00--24) • *mm* is minutes (00--59) • *ss* is seconds (00--59) • *cc* is century • *yy* is year (00--99)

Examples: Get_Current_Date_Time

Because no input data is required to use this component, just type into the calling rule:

```
use component GET_CURRENT_DATE_TIME
```

In this example, the current date and time are sent to the fields in the output view.

```
map CURRENT_TIME of GET_CURRENT_DATE_TIME_O to
    TIME of SELECT_CUSTOMER_O
map CURRENT_DATE of GET_CURRENT_DATE_TIME_O to
    DATE of SELECT_CUSTOMER_O
use component GET_CURRENT_DATE_TIME
```

In this example, the current time is then mapped to a field on the Select_Customer window.

```
map CURRENT_DATE of GET_CURRENT_DATE_TIME_O to
    DATE of SELECT_CUSTOMER
use component GET_CURRENT_DATE_TIME
```

Get_Elevator_Position

This section presents the purpose, the syntax, the usage, and how the Get_Elevator_Position component is supported. See Example: Get_Elevator_Position for an usage example.

Purpose

Use this component to obtain the occurrence number of the current elevator (scroll) position. The occurrence number is a virtual number if a virtual list box is defined. This component determines how far and in which direction an end user has scrolled the values in a multicolumn list box (MCLB). If the view name is invalid, a zero value is returned to the Elevator_Position field of the output view.

⚠ You must have conversed the window containing the list box at least once for this component to work.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Get_Elevator_Position component.

Get_Elevator_Position component syntax

Name	`Get_Elevator_Position`
System Identifier	`CGETELV`
Input View Name	`Get_Elevator_Position_I`
Input View Field	`View_Long_Name character(30)`
Output View Name	`Get_Elevator_Position_O`
Output View Field	`Elevator_Position integer(15)` Occurrence associated with current elevator position.

Example: Get_Elevator_Position

This example gets the elevator position.

```
map 'CUSTOMER_CDV'to VIEW_LONG_NAME of GET_ELEVATOR_POSITION_I
use component GET_ELEVATOR_POSITION

map ELEVATOR_POSITION of GET_ELEVATOR_POSITION_O
    to ELEVATOR_POSITION of NC_TABLECOMPONENT_WD of
       NC_TABLECOMPONENT_W
```

Get_First_Visible_Occurrence

This section presents the purpose, the syntax, the usage, and how the Get_First_Visible_Occurence component is supported. See Example: Get_First_Visible_Occurrence for an usage example.

Purpose

Use this component to obtain the occurrence number of the first visible occurrence in an occurring view or a multicolumn list box (MCLB). If the view name is invalid, or if the view name is not linked to a MCLB, a zero value is returned to the Field_Occur field. This component works only with MCLBs.

⚠ You must have conversed the window containing the multicolumn list box at least once for this component to work.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Get_First_Visible_Occurrence component.

Get_First_Visible_Occurrence component syntax

Name	`Get_First_Visible_Occurrence`
System Identifier	`CGETFST`
Input View Name	`Get_First_Visible_Occurrence_I`
Input View Field	`View_Long_Name character(30)`
Output View Name	`Get_First_Visible_Occurrence_O`
Output View Field	`Field_Occur integer(15)` Occurrence number.

Example: Get_First_Visible_Occurrence

This example gets the first occurrence number.

```
map 'MCLB_OCC' to VIEW_LONG_NAME of GET_FIRST_VISIBLE_OCCURRENCE_I
use component GET_FIRST_VISIBLE_OCCURRENCE
map FIELD_OCCUR of GET_LAST_VISIBLE_OCCURRENCE_O to
    FIELD_OCCUR of NC_TABLECOMPONENT_WD of NC_TABLECOMPONENT_W
map 'CUSTOMER_CDV'to VIEW_LONG_NAME of GET_ELEVATOR_POSITION_I
use component GET_ELEVATOR_POSITION
```

Get_Full_User_Identity

This section presents the purpose, the syntax, the usage, and how the Get_Full_User_Identity component is supported. See Examples: Get_Full_User_Identity for an usage example.

Purpose

Use this component to obtain the workstation identifier and user (log-on) identifier. This component returns as much information as it can find on a workstation. Thus, if security was not run (if a log-in screen asking for user name and password was not displayed), the user identifier field is blank. If, in addition, there is no workstation identifier, then that field is blank as well. The value returned for the user identifier is the identifier entered by a user in the security window when signing on.
For a cooperative-processing application, this component does not work unless CICS or IMS security is activated.

The function of this component is identical to the older Get_User_Workstation_ID component. However, this component has longer fields in the output view to provide for future extensions. The older component continues to be supported to accommodate applications that are being upgraded. New applications should use the Get_Full_User_Identity component.

C Support

In C applications, the current workstation identifier is retrieved from the value of the WORKSTATION_ID setting in the AE Runtime section of the system initialization file (hps.ini). Before using this component, this setting in the hps.ini file must be assigned a string value.
A user id is only returned if the GET_USERID_FROM_DNA setting in the [AE Runtime] section of hps.ini has been set to TRUE, and security has been called, i.e. that an authentication exit (DNA_AUTHENT_EXIT of dna.ini) has been called.

C# Support

This component is supported for C# client. The machine host name is retrieved for C# applications.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients. For Java applications, the current workstation identifier is the host name of the machine.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. For 3270 Converse applications, this component returns the user identifier and *netname* for the mainframe session. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Get_Full_User_Identity component.

Get_Full_User_Identity component syntax

Name	`Get_Full_User_Identity`
System Identifier	`CGETUIF`
Input View Name	This component does not have an input view.
Output View Name	`Get_Full_User_Identity_O`
Output View Field	`Workstation_ID_30 character(30)` Identifier of the workstation on which the component is being run.
Output View Field	`User_ID_30 character(30)` Identifier of the user who has logged on to the workstation on which the component is being run.

Examples: Get_Full_User_Identity

This example maps the workstation identifier to the variable THIS_PC_ID.

```
map WORKSTATION_ID_30 of GET_FULL_USER_IDENTITY_O to THIS_PC_ID
use component GET_FULL_USER_IDENTITY
```

This example maps the identifier of the user who logged on to the workstation to the variable USER_NAME.

```
map USER_ID_30 of GET_FULL_USER_IDENTITY_O to USER_NAME
use component GET_FULL_USER_IDENTITY
```

Get_Last_Visble_Occurence

This section presents the purpose, the syntax, the usage, and how the Get_Last_Visible_Occurence component is supported. See Example: Get_Last_Visible_Occurrence for an usage example.

Purpose

Use this component to obtain the occurrence number of the last visible occurrence of a field in an occurring view or a multicolumn list box (MCLB). If the view name is invalid, or if the view name is not linked to a MCLB, a zero value is returned to the Field_Occur field.

⚠ You must have conversed the window containing the multicolumn list box at least once for this component to work.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Get_Last_Visible_Occurrence component.

Get_Last_Visible_Occurrence component syntax

Name	`Get_Last_Visible_Occurrence`
System Identifier	`CGETLST`
Java Support	Yes
Input View Name	`Get_Last_Visible_Occurrence_I`
Input View Field	`View_Long_Name character(30)`
Output View Name	`Get_Last_Visible_Occurrence_O`
Output View Field	`Field_Occur integer(15)` Occurrence number.

Example: Get_Last_Visible_Occurrence

This example gets the last occurrence number.

```
map 'NC_TABLE_SL' to VIEW_LONG_NAME of GET_LAST_VISIBLE_OCCURRENCE_I
use component GET_LAST_VISIBLE_OCCURRENCE
map FIELD_OCCUR of GET_LAST_VISIBLE_OCCURRENCE_O to
    FIELD_OCCUR of NC_TABLECOMPONENT_WD of NC_TABLECOMPONENT_W
```

Get_Listbox_Window_Sizes

This section presents the purpose, the syntax, the usage, and how the Get_Listbox_Window_Sizes component is supported. See Example: Get_Listbox_Window_Sizes for an usage example.

Purpose

Use this component to obtain the number of lines visible in the multicolumn list box (MCLB) window (the number of visible occurrences) and the total number of scrollable occurrences available in the MCLB view when given the name of a multicolumn list box view. Although the output view does not contain a Return_Code field, if Visible_Occurs returns a value of 0, the attempt to use the component failed. If the specified view name is invalid or if the view is not linked to a multicolumn list box, the component fails. This component works only with multicolumn list boxes.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Get_Listbox_Window_Sizes component.

Get_Listbox_Window_Sizes component

Name	`Get_Listbox_Window_Sizes`
System Identifier	`CLSTWSZ`
Input View Name	`Get_Listbox_Window_Sizes_I`
Input View Field	`View_Long_Name character(30)`
Output View Name	`Get_Listbox_Window_Sizes_O`
Output View Field	`Visible_Occurs integer(15)` The number of visible occurrences.
Output View Field	`Scrollable_Occurs integer(15)` The number of occurrences in the view that can be scrolled without setting off an out-of-range condition.

Example: Get_Listbox_Window_Sizes

This example gets the first occurrence number.

```
map 'NC_TABLE_SL' to VIEW_LONG_NAME of GET_LISTBOX_WINDOW_SIZES_I
use component GET_LISTBOX_WINDOW_SIZES
map VISIBLE_OCCURS of GET_LISTBOX_WINDOW_SIZES_O to
    L_VISIBLE_OCCURS
map SCROLLABLE_OCCURS of GET_LISTBOX_WINDOW_SIZES_O
    to L_SCROLLABLE_OCCURS
```

Get_Menu_Mode_By_ID

This section presents the purpose, the syntax, the usage, and how the Get_Menu_Mode_By_ID component is supported. Example: Get_Menu_Mode_By_ID for an usage example.

Purpose

Use this component to find out the status of either of the two independent properties of a menu choice: whether the menu choice is enabled or whether the menu choice is checked. With each use of this component you can determine the status of one these two properties. The Enabled property refers to whether the menu item is enabled (active and selectable) or disabled (inactive and grayed out). The Checked property refers to whether the menu item is checked (has a check mark next to it) or not (has no check mark next to it). This component is the "get" version of Set_Menu_Mode_By_ID.
The two properties are independent. A menu choice can be enabled and checked, enabled and unchecked, disabled and checked, or disabled and unchecked. To find out that status of both properties, you must use two calls — one to find out if it is enabled and one to find out if it is checked.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Get_Menu_Mode_By_ID component.

Get_Menu_Mode_By_ID component syntax

Name	`Get_Menu_Mode_by_ID`
System Identifier	`CGETMMI`
Input View Name	`Get_Menu_Mode_by_ID_I`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the menu choice
Input View Field	`HPS_MItem_Attribute integer(15)` The property of the menu choice that you want to check: • 0 = Enabled - whether enabled (active) or disabled (grayed out) • 1 = Checked - whether it has a check mark next to it or not This field is attached to the system set MENU_ITEM_ATTRIBUTES (SMIATTR).
Output View Name	`Get_Menu_Mode_by_ID_O`
Output View Field	`HPS_MItem_State integer(15)` The status of the property specified on the input view. (Only one property can be checked per call of this component.) • 0 = For Enabled, this means the menu choice is disabled (grayed out); For Checked this means the menu choice is not checked. • 1 = For Enabled, this means the menu choice is enabled (active); For Checked this means the menu choice has a check mark next to it. This field is attached to the system set MENU_ITEM_STATE (SMIST).
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: Get_Menu_Mode_By_ID

This part of the example determines whether the **Menu Item 2** choice is enabled. (It does not determine whether the choice is checked.)

```
map 'Menu Item 2' to PS_ITEM_ID of GET_MENU_MODE_BY_ID_I
map 0 to HPS_MITEM_ATTRIBUTE of GET_MENU_MODE_BY_ID_I
use component GET_MENU_MODE_BY_ID
```

This part of the example maps the returned value of the Enabled property status to the variable `MENU_STATE`. In this example, if 0 is returned, then **Menu Item 2** is disabled or grayed out; if 1 is returned, it is enabled and active. This does not affect whether or not the menu choice is checked.

```
map HPS_MITEM_STATE of GET_MENU_MODE_BY_ID_O to MENU_STATE
use component GET_MENU_MODE_BY_ID
```

Get_Selected_Field

This section presents the purpose, the syntax, the usage, and how the Get_Selected_Field component is supported. See Examples: Get_Selected_Field for an usage example.

Purpose

Use this component to determine the screen fields users select in a list box. You can use this component iteratively to find all the selected screen fields. The component returns the location (view name, field name, and occurrence number) of the first, next, or specified selected field. If a view name is specified, but a field name is not, the component searches for any selected field within that view. Also see HPS_Set_Selected_Field.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

⚠ This component is not supported when a blank view name is given.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for information on other components.
For 3270 Converse, this component applies to all fields---not only those in a list box. However, you should use this component only once before a converse because only one field can be selected in 3270 Converse. For 3270 Converse applications executing on the mainframe, the selected field is the field containing the cursor when the window was last conversed. The options from the FIELD_OCCUR parameter are not supported.

Syntax

Here is the syntax of Get_Selected_Field component.

Get_Selected_Field component syntax

Name	`Get_Selected_Field`
System Identifier	`CGETFLD`
Input View Name	`Get_Selected_Field_I`
Input View Field	`View_Long_Name character(30)` The view that includes the field mapped to `Field_Long_Name`
Input View Field	`Field_Long_Name character(30)` The name of the field in the hierarchy diagram to which the field in the window is linked

Input View Field	`Field_Occur integer(15)` Occurrence number of the altered field. This field is attached to the system set SEARCH_CRITERIA (SSEARCH), which has the following values: • -1 = Return first selected field • 0 = Return next selected field • *n* = Return only the selected field that is the *n* th occurrence in a list box
Output View Name	`Get_Selected_Field_O`
Output View Field	`View_Long_Name character(30)`
Output View Field	`Field_Long_Name character(30)`
Output View Field	`Field_Occur integer(15)` This parameter directs the component to return the first selected field found, the next selected field, or a field with a particular occurrence number in an occurring view such as a list box. If the virtual size for the table is enabled (with Set_Virtual_Listbox_Size), then the value returned is the virtual index.
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Usage

This component returns the view name, field name, and field occurrence of the selected screen field to the calling rule when this component is successful. When no more selected fields are found, the Return_Code field is set to failure. An iterative call to this component loops through all selected fields until a selected field is deselected by mouse action or cleared by a call to Clear_Selected_Fields.
The hps.ini parameter LISTBOX_SEL_ORDER governs selection of the fields for this component. This parameter can have one of two values, either NATURAL or SELECTION_ORDER (the default). If the SELECTION_ORDER value is used, the fields are returned in the order selected. If the NATURAL value is used, the fields are returned in the order they appear in the multicolumn list box, regardless of the order in which they were selected.

⚠ Except for 3270 Converse, this component applies only to fields in a list box. This is a change from earlier versions of the product.

Examples: Get_Selected_Field

This example specifies CUSTOMER as the view on which to search for the selected fields.

```
map 'CUSTOMER' to VIEW_LONG_NAME of GET_SELECTED_FIELD_I
```

This example specifies CUSTOMER_NAME as the selected field to search.

```
map 'CUSTOMER_NAME' to FIELD_LONG_NAME of GET_SELECTED_FIELD_I
```

This example maps the value NEXT_SELECTION of the system set Search_Criteria to the input view, which causes the search to continue to the next instance of the selected field.

```
map NEXT_SELECTION in SEARCH_CRITERIA to FIELD_OCCUR of
    GET_SELECTED_FIELD_I
```

If a selected field is found, this maps the value of the field CUSTOMER_NAME in the occurring view CUSTOMER (the FIELD_OCCUR occurrence of the view CUSTOMER) to the field CUSTOMER_NAME of the input view CUSTOMER_BILLING_INFO_I.

```
converse window SELECT_CUSTOMER
use component GET_SELECTED_FIELD
if RETURN_CODE = SUCCESS in RETURN_CODES
   map CUSTOMER_NAME of CUSTOMER (FIELD_OCCUR of
      GET_SELECTED_FIELD_O) to
         CUSTOMER_NAME of CUSTOMER_BILLING_INFO_I
   use rule CUSTOMER_BILLING_INFO
else
   use component SET_WINDOW_MESSAGE
```

Get_Text_Message

This section presents the purpose, the syntax, the usage, and how the Get_Text_Message component is supported. See Example: Get_Text_Message for an usage example.

Purpose

Use this component to read a reference file (a set denoted by the REF extension) to obtain a particular message based on the TEXT code the input view supplies. The component returns up to four lines of text message.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Get_Text_Message component.

Get_Text_Message component syntax

Name	`Get_Text_Message`
System Identifier	`CGETMSG`
Input View Name	`Get_Text_Message_I`
Input View Field	`Message_Set_Name character(8)` The implementation name of the reference file without the REF extension
Input View Field	`Text_Code integer(15)` A number indicating which message to use from the error set
Output View Name	`Get_Text_Message_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.
Output View Field	`Txt_Arg1 character(35)` First line of text in output message
Output View Field	`Txt_Arg2 character(35)` Second line of text in output message
Output View Field	`Txt_Arg3 character(35)` Third line of text in output message

Output View Field	`Txt_Arg4 character(35)` Fourth line of text in output message

Usage

See the description for either Set_Field_Message or Set_Window_Message for information on how place holders (% and @) are used for substitution in the reference file. If you use this component to read a reference file that contains one of these place holders, the place holder is returned in the corresponding Txt_Arg field.
There is an HPS.INI setting SKIP_BLANK_LINES_IN_SET that allows the C client to recognize blank lines in the set text when using this component. The ini setting is located in the [AE Runtime] section of the HPS.ini file. The default is TRUE. When set to FALSE the component returns blank lines from the set.

Example: Get_Text_Message

This example specifies the set with the implementation name ERRMSGS as the set from which to retrieve messages.

```
map 'ERRMSGS' to MESSAGE_SET_NAME of GET_TEXT_MESSAGE_I
```

This example maps the value CUST_NOT_ON_FILE of the set ERROR_MESSAGES to the input view, specifying the text message corresponding to that value to be displayed. Note that this must be the set whose implementation name was mapped to Message_Set_Name above.

```
map CUST_NOT_ON_FILE of ERROR_MESSAGES to TEXT_CODE of
    GET_TEXT_MESSAGE_I
```

Get_User_Workstation_ID

This section presents the purpose, the syntax, the usage, and how the Get_Uder_Workstation_ID component is supported. See Examples: Get_User_Workstation_ID for an usage example.

Purpose

Use this component to obtain the workstation identifier and user (log-on) identifier.
This component returns as much information as it can find on a workstation. Thus, if security was not run (if a log-in screen asking for user name and password was not displayed), the user identifier field is blank. If, in addition, there is no workstation identifier, then that field blank as well. The value returned for the user identifier is the identifier entered by a user in the security window when signing on.
For a cooperative-processing application, this component does not work unless CICS or IMS security is activated.

The function of this component is identical to the newer Get_Full_User_Identity component. However, the new component has longer fields in the output view to provide for future extensions. This component continues to be supported to accommodate applications that are being upgraded. New applications should use Get_Full_User_Identity component.

In C applications, the current workstation identifier is retrieved from the value of the WORKSTATION_ID setting in the AE Runtime section of the system initialization file (hps.ini). Before using this component, this setting in the hps.ini file must be assigned a string value. The user identifier is retrieved from the value typed in when security was run.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information. For 3270 Converse applications, this component returns the user identifier and *netname* for the mainframe session.

Syntax

Here is the syntax of Get_User_Workstation_ID component.

Get_User_Workstation_ID component syntax

Name	`Get_User_Workstation_ID`
System Identifier	`CGETUWI`
Input View Name	This component does not have an input view.
Output View Name	`Get_User_Workstation_ID_O`
Output View Field	`Workstation_ID character(16)` Identifier of the workstation on which the component is being run
Output View Field	`User_Identity character(7)` Identifier of the user who has logged on to the workstation on which the component is being run

Examples: Get_User_Workstation_ID

This example maps the workstation ID to the variable `SELECT_THIS_PC`.

```
map WORKSTATION_ID of GET_USER_WORKSTATION_ID_O to SELECT_THIS_PC
    use component GET_USER_WORKSTATION_ID
```

This example maps the identifier of the user who logged on to the workstation to the variable `USER_NAME`.

```
map USER_IDENTITY of GET_USER_WORKSTATION_ID_O to USER_NAME
```

HPS_Event_Post_to_Child

This section presents the purpose, the syntax, the usage, and how the HPS_Event_Post_to_Child component is supported. See Example: HPS_Event_Post_to_Child for an usage example.

Purpose

Use this component, in a parent (primary) rule, to post an event to a child rule (or multiple child rules). Use this component with event-driven processing and modeless windows. Specify the child (secondary) rule by the combination of Event_Dest and Event_Qualifier.

⚠ The use of this component is discussed in detail in *Developing Applications Guide* .

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is not supported for C# client.

Syntax

Here is the syntax of HPS_Event_Post_to_Child component.

HPS_Event_Post_to_Child component syntax

Name	`HPS_Event_Post_to_Child`
System Identifier	`CPOSTC`
Input View Name	`HPS_Event_Post_to_Child_I`

Input View Field	`Event_Name character(30)` Name of event to post
Input View Field	`Event_Dest character(30)` Name of the child rule receiving the event
Input View Field	`Event_Qualifier character(30)` Optional identifier used to distinguish different instances of the same rule
Input View Field	`Event_View character(30)` Optional name of work view being sent, which (if specified) must be connected to both the sending and receiving rules and cannot be the name of the pre-defined system view HPS_EVENT_VIEW
Input View Field	`Event_Param character(256)` Additional information
Output View Name	`HPS_Event_Post_to_Child_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: HPS_Event_Post_to_Child

This example posts an event to a child rule.

```
map 'Query and focus' to EVENT_NAME of HPS_EVENT_POST_TO_CHILD_I
map 'OWNERSHIP_QRY_DIS' to EVENT_DEST of HPS_EVENT_POST_TO_CHILD_I
use component HPS_EVENT_POST_TO_CHILD
```

To receive the event on the child, use either a converse loop, a proc for ConverseEvent or a proc for PostEvent. Refer to the *Developing Applications Guide* and the *ObjectSpeak Reference Guide* for more information on using these procedures.

HPS_Event_Post_to_Parent

This section presents the purpose, the syntax, the usage, and how the HPS_Event_Post_to_Parent component is supported. See Example: HPS_Event_Post_to_Parent for an usage example.

Purpose

Use this component, in a child (secondary) rule, to post an event to its parent (primary) rule. This component is used with event-driven processing and modeless windows. The parent rule is the one that did the DETACH.

⚠ The use of this component is discussed in detail in *Developing Applications Guide*.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is not supported for C# client.

Syntax

Here is the syntax of HPS_Event_Post_to_Parent component.

HPS_Event_Post_to_Parent component syntax

Name	`HPS_Event_Post_to_Parent`
System Identifier	`CPOSTP`

Input View Name	`HPS_Event_Post_to_Parent_I`
Input View Field	`Event_Name character(30)` Name of event to post
Input View Field	`Event_View character(30)` The name of the work view being sent. This must be connected to both the sending and receiving rules and cannot be the name of the pre-defined system view HPS_EVENT_VIEW.
Input View Field	`Event_Param character(256)` Additional information
Output View Name	`HPS_Event_Post_to_Parent_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: HPS_Event_Post_to_Parent

This example posts an event to a parent rule.

```
map 'Query and focus' to EVENT_NAME of HPS_EVENT_POST_TO_PARENT_I
map 'OWNERSHIP_QRY_DIS' to EVENT_VIEW of HPS_EVENT_POST_TO_PARENT_I
use component HPS_EVENT_POST_TO_PARENT
```

To receive the event on the parent, use either a converse loop, a proc for ConverseEvent or a proc for PostEvent. Refer to the *Developing Applications Guide* and the *ObjectSpeak Reference Guide* for more information on using these procedures.

HPS_Get_Activate

This section presents the purpose, the syntax, the usage, and how the HPS_Get_Activate component is supported. See Example: HPS_Get_Activate for an usage example.

Purpose

Use this component to determine if this is the active process. When the window of this process is active, the process is active.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.

C# Support

This component is not supported for C# client.

Syntax

Here is the syntax of HPS_Get_Activate component.

HPS_Get_Activate component syntax

Name	`HPS_Get_Activate`
System Identifier	`CGETACT`
Input View Name	This component does not have an input view.
Output View Name	`HPS_Get_Activate_O`
Output View Field	`HPS_Active_State integer(15)` This field is attached to the system set HPS_ACTIVE_STATES (SACSTAT), which has the following values: 1 = Active 0 = Inactive

Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: HPS_Get_Activate

This example shows that if the value in ACTIVE_WINDOW is 1, then the window of this process is currently active.

```
map HPS_ACTIVE_STATE of HPS_GET_ACTIVATE_O to ACTIVE_WINDOW
use component HPS_GET_ACTIVATE
```

HPS_Get_Environment

This section presents the purpose, the syntax, the usage, and how the HPS_Get_Environment component is supported. See Example: HPS_Get_Environment for an usage example.

Purpose

Use this component to obtain the execution environment.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

OpenCOBOL for Unix and Java Batch Support

This component is supported for generation of OpenCOBOL for OpenCOBOL for Unix and Java Batch. It sets EXECMODE of the output view to "HP-UX" upon return.

Syntax

Here is the syntax of HPS_Get_Environment component.

HPS_Get_Environment component syntax

Name	`HPS_Get_Environment`
System Identifier	`CGETENV`
Input View Name	This component has no input view.
Output View Name	`HPS_Get_Environment_O`

Output View Field	`Execmode character(6)` The execution environment, with possible values of: <ul><li>AIX</li><li>BATCH (on mainframe)</li><li>CICS (on mainframe)</li><li>JVM (for Java client/server)</li><li>HP-UX</li><li>HTML (for thin client)</li><li>IMS (on mainframe)</li><li>NCR</li><li>PC DOS (Windows)</li><li>SUNOS</li><li>UNKNOW</li><li>WINNT</li></ul>

Example: HPS_Get_Environment

This example shows that the value in EXEC_ENVIRONMENT indicates the execution environment.

```
map EXECMODE of HPS_GET_ENVIRONMENT_O to EXEC_ENVIRONMENT
use component HPS_GET_ENVIRONMENT
```

HPS_Get_MinMax

This section presents the purpose, the syntax, the usage, and how the HPS_Get_MinMax component is supported. See Example: HPS_Get_MinMax for an usage example.

Purpose

Use this component to obtain the state (minimized, maximized, or normal) of the window of this process.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Get_MinMax component.

HPS_Get_MinMax component syntax

Name	`HPS_Get_MinMax`
System Identifier	`CGETMMX`
Input View Name	This component does not have an input view.
Output View Name	`HPS_Get_MinMax_O`
Output View Field	`HPS_Window_State integer(15)` This field is attached to the system set HPS_WINDOW_STATES (SWSTATE), which has the following values: <ul><li>0 = Minimized</li><li>1 = Maximized</li><li>2 = Normal</li></ul>
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: HPS_Get_MinMax

In this example, the value in WINDOW_SIZE indicates whether the window of this process is minimized, maximized, or normal.

```
map HPS_WINDOW_STATE of HPS_GET_MINMAX_O to WINDOW_SIZE
use component HPS_GET_MINMAX
```

HPS_Get_Window_Handle

This section presents the purpose, the syntax, the usage, and how the HPS_Get_Window_Handle component is supported. See Examples: HPS_Get_Window_Handle for an usage example.

Purpose

Use this component to return the system window handle for a client runtime window or one of the objects on the window. This is an immediate component that returns the window handle for a client runtime window when the input field HPS_Item_ID is blank. To retrieve the window handle for a specific control, enter the system identifier (HPS ID) of an object in the input field HPS_Item_ID.
The component is only supported for the client runtime on the Windows workstation with a rule that converses windows. We do not recommend using the window handle to alter client runtime applications.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Get_Window_Handle component.

HPS_Get_Window_Handle component syntax

Name	`HPS_Get_Window_Handle`
System Identifier	{CWINHD}}
Input View Name	`HPS_Get_Window_Handle_I`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPS ID) of a control on the panel.
Output View Name	`HPS_Get_Window_Handle_O`
Output View Field	`Window_Handle integer(31)` The system window handle.
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Examples: HPS_Get_Window_Handle

In this example, the value returned is the system window handle for a client runtime window.

```
map '' to HPS_ITEM_ID of HPS_GET_WINDOW_HANDLE_I
use component HPS_GET_WINDOW_HANDLE
```

In this example, the value returned is the system window handle for an object on the client runtime window with the system identifier (HPS ID) of EDIT_HPS_ID.

```
map 'EDIT_HPS_ID' to HPS_ITEM_ID of HPS_GET_WINDOW_HANDLE_I
use component HPS_GET_WINDOW_HANDLE
```

HPS_Intercept_Events

This section presents the purpose, the syntax, the usage, and how the HPS_Intercept_Events component is supported. See Example: HPS_Intercept_Events for an usage example.

Purpose

Use this component to re-route an event from a detached rule to a child modal rule (and its window) in the detaching process rather than to the root rule (and its window). This component re-routes external events to the calling rule instead of the root rule of the detaching process. It continues to intercept events until the rule is exited.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Intercept_Events component.

HPS_Intercept_Events component syntax

Name	`HPS_Intercept_Events`
System Identifier	`CINTEVT`
Input View Name	This component does not have an input view.
Output View Name	`HPS_Intercept_Events_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: HPS_Intercept_Events

In this example, the calling rule now intercepts posted events. You can check the Return_Code field to make sure the call was successful.

```
use component HPS_Intercept_Events
```

HPS_Limit_Scroll_Size

This section presents the purpose, the syntax, the usage, and how the HPS_Limit_Scroll_Size component is supported. See Examples: HPS_Limit_Scroll_Size for an usage example.

Purpose

Use this component to decrease the scrollable area of a combo box or list box. The component cannot be used to enlarge an object's scrollable area, nor does it change the physical size of the list box or combo box. For example, a list box could have 10 visible rows and 20 possible scrollable rows. If the component were used to decrease the scrollable rows to 5, the list box would still display 10 rows, but only the first 5 would contain data, and the vertical scrollbar would be disabled.
In C, this is a deferred component .

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Limit_Scroll_Size component.

HPS_Limit_Scroll_Size component syntax

Name	`HPS_Limit_Scroll_Size`
System Identifier	`CLMTSCR`
Input View Name	`HPS_Limit_Scroll_Size_I (System ID: CLMTSCRI)`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the combo box or list box
Input View Field	`Number_Of_Records integer(15)` The number of scrollable records in the combo box or list box.character()
Output View Name	`HPS_Limit_Scroll_Size_O (System ID: CLMTSCRO)`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

This component may be used to limit dynamically the scrollable size of a single-column list box. This is useful if a list box occurs, for example, 100 times, but only the first 15 occurrences are populated with data. The user can execute this component with the number of records equal to 15, and it is possible to scroll through the first fifteen rows of the list box.

If the component is executed with the number of records equal to -1, the scrollable size is made equal to the number of repository occurrences. The scrollable size may be made less than or equal to the number of repository occurrences, but not greater than this number.

⚠ This component is *not* intended to provide smooth scrolling support for list boxes.

Examples: HPS_Limit_Scroll_Size

This example limits the scrolling space in the list box associated with the combo box.

```
map COMBOBOX to HPS_ITEM_ID
map 10 to NUMBER_OF_RECORDS
use component HPS_LIMIT_SCROLL_SIZE
```

This example limits the scrollable size of the list box specified by LIST_3 to 20.

```
map 20 to NUMBER_OF_RECORDS of HPS_LIMIT_SCROLL_SIZE_I
map 'LIST_3' to HPS_ITEM_ID of HPS_LIMIT_SCROLL_SIZE_I
use component HPS_LIMIT_SCROLL_SIZE
```

HPS_Set_Activate

This section presents the purpose, the syntax, the usage, and how the HPS_Set_Activate component is supported. See Example: HPS_Set_Activate for an usage example.

Purpose

Use this component, when called by a rule, to make active the next window conversed within the process. That is, it brings that window to the top of all other windows and gives it the focus. When used with an HPS_Active_State of Inactive, this component causes the next window conversed to come to the top in an inactive state. If the window conversed appears on top of the active window, it drops back behind the active window.
In C, this is a deferred component.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.
In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Set_Activate component.

HPS_Set_Activate component

Name	`HPS_Set_Activate`
System Identifier	`CSETACT`
Input View Name	`HPS_Set_Activate_I`
Input View Field	`HPS_Active_State integer(15)` This field is attached to the system set HPS_ACTIVE_STATES (SACSTAT), which has the following values: 1 = Active 0 = Inactive
Output View Name	`HPS_Set_Activate_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: HPS_Set_Activate

In this example, the next window conversed becomes the active window.

```
map 1 to HPS_ACTIVE_STATE of HPS_SET_ACTIVATE_I
use component HPS_SET_ACTIVATE
```

HPS_Set_Bitmap_File

This section presents the purpose, the syntax, the usage, and how the HPS_Set_Bitmap_File component is supported. See Example: HPS_Set_Bitmap_File for an usage example.

Purpose

Use this component, when called by a rule, to change the bitmap file associated with a bitmap entity on a window. This component allows bitmaps

specified at design time (with Window Painter) to be changed dynamically at run time.
In C, this is a deferred component .

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.
In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information. The bitmap file should be placed into the directory included in the classpath.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Set_Bitmap_File component.

HPS_Set_Bitmap_File component

Name	`HPS_Set_Bitmap_File`
System Identifier	`CSETBMP`
Input View Name	`HPS_Set_Bitmap_File_I`
Input View Field	`HPS_Bitmap_Name character(50)` The name of the bitmap file (BMP) containing the bitmap to be displayed. The bitmap file must be in the directory specified by the BITMAP_DIR variable in the AE Runtime section of the system initialization file (hps.ini), or if the bitmap file exists in a different directory, it must be fully qualified.
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the bitmap entity that is to be changed dynamically
Output View Name	`HPS_Set_Bitmap_File_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

If you pass a null bitmap file name to the component, it passes a return code of 1 to the rule, and the previous bitmap is redrawn when the window is conversed.
If you pass an invalid bitmap file name, the component passes a return code of 1 and does not display a bitmap when the window is conversed.

Example: HPS_Set_Bitmap_File

This example has a bitmap file, tools.bmp, associated with the bitmap entity whose identifier is 'ID00102'. The bitmap file replaces whatever file was previously associated with the bitmap entity.

```
map 'Tools.bmp' to HPS_BITMAP_NAME of HPS_SET_BITMAP_FILE_I
map 'ID00102' to HPS_ITEM_ID of HPS_SET_BITMAP_FILE_I
use component HPS_SET_BITMAP_FILE
```

HPS_Set_Cursor_By_ID

This section presents the purpose, the syntax, the usage, and how the HPS_Set_Cursor_By_ID component is supported. See Examples: HPS_Set_Cursor_By_ID for an usage example.

Purpose

Use this component to place the focus on a particular Window Painter object (that is, run-time control) based on its system identifier.
In C, this is a deferred component .

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.
In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Set_Cursor_By_ID component.

HPS_Set_Cursor_By_ID component syntax

Name	`HPS_Set_Cursor_By_ID`
System Identifier	`CCRSRID`
Input View Name	`HPS_Set_Cursor_By_ID_I (System ID: CCRSRIDI)`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the field where cursor should be placed or the system identifier of an object on the window
Input View Field	`Field_Occur integer(15)`
Output View Name	`HPS_Set_Cursor_By_ID_O (System ID: CCRSRIDO)`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

This deferred component is used to position the cursor in a specific editable or unprotected object when the window is re-conversed. The field selected is highlighted, enabling a user to enter text immediately and overwrite any text already in the field. This component works with the edit field, multi-line edit field, combo box, list box, checkbox, radio button and push button objects, as well as the columns of a multicolumn list box (MCLB). The component cannot be used to select a protected field or object.
The component requires the system identifier (HPSID) of the object that is to be selected.
The optional FIELD OCCUR value can be used to position the cursor on different rows of a list box or multicolumn list box object. The default value of FIELD_OCCUR is 0.

Examples: HPS_Set_Cursor_By_ID

This example sets the focus to the push button whose system identifier (HPSID) is EXITPUSH.

```
map 'EXITPUSH' to HPS_ITEM_ID of HPS_SET_CURSOR_BY_ID_I
use component HPS_SET_CURSOR_BY_ID
```

This example sets the cursor in an edit field.

```
map EDITFIELD to HPS_ITEM_ID
use component HPS_SET_CURSOR_BY_ID
```

HPS_Set_Help_Topic

This section presents the purpose, the syntax, the usage, and how the HPS_Set_Help_Topic component is supported. See Example: HPS_Set_Help_Topic for an usage example.

Purpose

Use this component to set a default help topic for all objects in a window. Normally, when an end user selects help for an object within a window, you use the *Show_Help_Topic* component to display the help text associated with the object's system identifier (HPSID). This component allows you to override the system identifier and display the same default help topic no matter which object in the window that help is called from. The keyword specified in the HELP_KEYWORD field must be in the same help file set by the last use of SET_HELP_FILE_NAME.

In C, this is a deferred component.

⚠ Use this component only with the native GUI help files associated with your application. Do not use this component if you are using built-in help features of AppBuilder.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.
In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Set_Help_Topic component.

HPS_Set_Help_Topic component syntax

Name	`HPS_Set_Help_Topic`
System Identifier	`CSETHLT`
Input View Name	`Set_Help_Topic_I`
Input View Field	`Window_Long_Name character(30)`
Input View Field	`Help_Keyword character(50)` Keyword of topic to be displayed.
Output View Name	`Set_Help_Topic_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: HPS_Set_Help_Topic

If CUSTOMER_DETAILS is the next window conversed, any help called references the KEYWORD topic. (If CUSTOMER_DETAILS is not the next window conversed, the component has no effect).

```
map 'CUSTOMER_DETAILS' to WINDOW_LONG_NAME of HPS_SET_HELP_TOPIC_I
map 'KEYWORD' to HELP_KEYWORD of HPS_SET_HELP_TOPIC_I
use component HPS_SET_HELP_TOPIC
```

HPS_Set_HTML_File

This section presents the purpose, the syntax, the usage, and how the HPS_Set_HTML_File component is supported. See Example: HPS_Set_HTML_File for an usage example.

Purpose

This component merges the specified HTML file with the window content and it is re-displayed when the event procedure is completed. This HTML file cannot be a complete page---that is, it cannot contain HTML, HEAD, or BODY tags. You can insert additional pieces of HTML code to be displayed at run time. For example, by displaying dynamically generated HTML, your Web program can display error messages stored in a database.

Java Support

This component is supported for thin (HTML) clients but not thick (Java) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Set_HTML_File component.

HPS_Set_HTML_File component syntax

Name	`HPS_Set_HTML_File`
System Identifier	`CSETHTM`
Input View Name	`HPS_Set_HTML_File_I`
Input View Field	`HPS_HTML_File_Name character(260)` The name of the file containing the HTML fragment to be displayed at run time, up to 260 characters. Using a forward slash, "/", for the file and directory separator ensures that your code is portable across platforms.

Input View Field	`HPS_HTML_Macro character(30)`
	The name of the macro, up to 30 characters.
	When the following string is found in the HTML file, it is replaced with the contents of the file specified in HPS_HTML_File_Name:
	`<LVEL_DYNAMIC name="macroname">`

Output View Name	`HPS_Set_HTML_File_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Usage

To use this component, first use the Repository Migration tool in the Repository Administration tool to import HPS_Set_HTML_File and HPS_Set_HTML_Fragment. For more information on Repository Administration and migrating files, refer to the *Personal and Workgroup Repository Administration Guide* .

Example: HPS_Set_HTML_File

This example uses a window, Window1, that displays dynamically generated HTML at run time. Also refer to the procedure in HPS_Set_HTML_Fragment.

1. Edit HTML for the window Window1, insert the following HTML comment where you want the dynamically generated HTML to appear at

run time:

```
<LVEL_DYNAMIC name="macroname">
```

where the macroname is the name used in the input view of the system component.

2. Save the HTML file for the Web page for Window1, close the editor, and commit the changes to the repository.
3. Use the component in the display rule.
4. Prepare your application, including the window Window1.
5. At run time, the HTML tag, LVEL_DYNAMIC is replaced with the HTML fragment corresponding to *macroname* .

Make sure that you can insert the dynamically generated HTML where you have indicated. Call the component each time the window is displayed, because returning control to the rule resets the HTML file names.

⚠ If you make any changes in Window Painter and then regenerate the HTML (with right-click HTML>Regenerate HTML), it overwrites any changes you have already made in your HTML editor. Selecting this option removes the HTML comment tags used by the component.

In the rule displaying window Window1, use this component as shown in the following code:

```
map 'macroname' to HPS_HTML_MACRO of HPS_SET_HTML_FILE_I
map 'filename' to HPS_HTML_FILE_NAME of HPS_SET_HTML_FILE_I
use component HPS_SET_HTML_FILE
```

filename can be either:

- An existing file that you refer to. Note, existing files can be stored in the repository as a component folder attached to the window. Attached files are prepared with the window to the `includes` subdirectory.
- A temporary file, created with user-written components, that contains the HTML fragment. Remember that your application should delete the temporary file after using it.

At run time, the HTML comment tag is replaced with the HTML fragment in the corresponding file.
Remember to do the following:

- Correctly specify the path for the file containing the HTML fragment.
- Enclose *macroname* and *filename* in quotation marks in the rule.
- Make sure the HTML file is a fragment rather than a complete page. That is, it cannot contain <HTML>, <HEAD>, or <BODY> tags.
- Make sure you associate a file with the macro. The macro has no effect if you do not associate a file with it.
- Remember the names and locations of the files, because you need to reinsert the macros if you later edit the window in Window Painter and regenerate the HTML code.
- Make sure your application deletes any temporary files it creates.

HPS_Set_HTML_Fragment

This section presents the purpose, the syntax, the usage, and how the HPS_Set_HTML_Fragment component is supported. See Example: HPS_Set_HTML_Fragment for an usage example.

Purpose

Use this component to display a dynamically generated HTML fragment (additional HTML code) stored as a character string within the rule that uses the component.

Java Support

This component is supported for thin (HTML) clients but not thick (Java) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Set_HTML_Fragment component.
HPS_Set_HTML_Fragment component syntax

Name	`HPS_Set_HTML_Fragment`
System Identifier	`CFRGHTM`
Input View Name	`HPS_Set_HTML_Fragment_I`

Input View Field	`HPS_HTML_Macro character(30)`
	The name of the macro, up to 30 characters.
	When the following string is found in the HTML file, it is replaced with the contents of the file specified in HPS_HTML_File_Name:
	`<LVEL_DYNAMIC name="macroname">`

Input View Field	`HPS_HTML_Fragment character(4000)`
Input View Field	`HPS_HTML_Fragment_Length integer(31)`
Input View Field	`HPS_HTML_Fragment_Append integer(15)` Use this to append more and display more than the 4,000 characters allowed in HPS_HTML_Fragment field. If APPEND=1, each new fragment is added to the end of the string. If APPEND=0, the fragment displayed is replaced each time the component is called. (The APPEND function does not apply if you use a different macroname to call the component.)
Output View Name	`HPS_Set_HTML_Fragment_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Usage

To use this component, first use the Migration Import tool in Freeway Explorer to import HPS_Set_HTML_File and HPS_Set_HTML_Fragment. For more information on Freeway Explorer and migrating files, refer to the *Personal and Workgroup Repository Administration Guide* .
Use this component to incorporate additional HTML fragments at run time.

Example: HPS_Set_HTML_Fragment

This example uses a window, Window1, that displays dynamically generated HTML at run time. Also refer to the procedure in HPS_Set_HTML_File.

1. Edit HTML for the window Window1, insert the following HTML comment where you want the dynamically generated HTML to appear at run time:

   ```
   <LVEL_DYNAMIC name="macroname">
   ```

   where the macroname is the name used in the input view of the system component.

2. Save the HTML file for the Web page for Window1, close the editor, and commit the changes to the repository.
3. Use the component in the display rule.
4. Prepare your application, including the window Window1.
5. At run time, the HTML tag, LVEL_DYNAMIC is replaced with the HTML fragment corresponding to *macroname* .

Make sure that you can insert the dynamically generated HTML where you have indicated. Call the component each time the window is displayed, because returning control to the rule resets the HTML file names.

⚠ If you make any changes in Window Painter and then regenerate the HTML (with right-click HTML>Regenerate HTML), it overwrites any changes you have already made in your HTML editor. Selecting this option removes the HTML comment tags used by the component.

In the rule conversing window Window1, use this component as shown in the following code:

```
map 'macroname' to HPS_HTML_MACRO of HPS_SET_HTML_FRAGMENT_I
map 'This is the character string to be displayed'
   to HPS_HTML_FRAGMENT of HPS_SET_HTML_FRAGMENT_I
map 40 to HPS_HTML_FRAGMENT_LENGTH of HPS_SET_HTML_FRAGMENT_I
map 0 to HPS_HTML_FRAGMENT_APPEND of HPS_SET_HTML_FRAGMENT_I
use component HPS_SET_HTML_FRAGMENT
```

At run time, the HTML comment tag is replaced with the character string mapped to the field HPS_HTML_Fragment.

In the rule, enclose the macroname and the character string mapped to the HTML fragment in quotation marks. If you call this component more than once using the same macroname, the field HPS_HTML_Fragment_Append can be used to display more than 4,000 characters. If APPEND=1, each new fragment is added to the end of the string. If APPEND=0, the fragment displayed is replaced each time the component is called. (The APPEND function does not apply if you use a different macroname to call the component.)

HPS_Set_MinMax

This section presents the purpose, the syntax, the usage, and how the HPS_Set_MinMax component is supported. See Example: HPS_Set_MinMax for an usage example.

Purpose

Use this component to minimize, maximize, or restore to normal the currently visible windows of this process. Minimize shows only one icon for all windows in this process. Normal restores windows to their original sizes and positions. Maximize maximizes the window as long as "Maximize box" is set to TRUE on the window property.
When this component is used in C environment, it doesn't minimizes, maximizes or restores the window until the next converse.
In C, this is a deferred component .

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.
With the Java client, this component works only in the current window. For HPS_Window_State input view field, the value of 1 (Maximize) does not work in JDK 1.3.1.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Set_MinMax component.

HPS_Set_MinMax component syntax

Name	`HPS_Set_MinMax`
System Identifier	`CSETMMX`
Input View Name	`HPS_Set_MinMax_I`
Input View Field	`HPS_Window_State integer(15)` This field is attached to the system set HPS_WINDOW_STATES (SWSTATE), which has the following values: 0 = Minimize, 1 = Maximize, 2 = Normal.
Output View Name	`HPS_Set_MinMax_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: HPS_Set_MinMax

This minimizes the currently visible windows of the active process.

```
map 0 to HPS_WINDOW_STATE of HPS_SET_MINMAX_I
use component HPS_SET_MINMAX
```

HPS_Set_Selected_Field

This section presents the purpose, the syntax, the usage, and how the HPS_Set_Selected_Field component is supported. See Example: HPS_Set_Selected_Field for an usage example.

Purpose

Use this component to select multiple rows or cells in either a list box or a multicolumn list box (MCLB) according to the selection type. For MCLBs, this component selects the entire rows or cells within the specified column, depending on the value of MCLB's Row Select property.

In C, this is a deferred component.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients, except the cell selection in HPS_Listbox_Cell_ID input view field is not supported.

In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Set_Selected_Field component.

HPS_Set_Selected_Field component syntax

Name	`HPS_Set_Selected_Field`
System Identifier	`CSETFLD`
Input View Name	`HPS_Set_Selected_Field_I`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the list box, MCLB or MCLB column.
Input View Field	`HPS_Listbox_Cell_ID character(50)` The system identifier (HPSID) of the MCLB column. You must specify this setting for MCLBs only when the Row Select property is set to FALSE, and the HPS_Item_ID contains the MCLB's HPSID rather than the column's HPSID. This setting is ignored for list boxes and MCLBs with the Row Select property set to TRUE.
Input View Field	`HPS_Field_Occur_Start integer(15)` First virtual occurrence number in range. Value must be greater than zero.
Input View Field	`HPS_Field_Occur_End integer(15)` Final virtual occurrence number in range and is not used for single row selection.
Output View Name	`HPS_Set_Selected_Field_O`

Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

The HPS_Item_ID field must contain the system identifier (HPSID) of the list box, MCLB or MCLB column. For MCLBs with the Row Select property set to FALSE, which allows individual cells to be selected, if the HPS_Item_ID field specifies the HPSID of an MCLB, the HPS_Listbox_Cell_ID field must contain the HPSID of a column within the MCLB. Alternatively, you may directly indicate the appropriate column by specifying it's HPSID in the HPS_Item_ID field. For MCLBs with the Row Select property set to TRUE, the HPS_Listbox_Cell_ID field is ignored, and specifying a column's HPSID in the HPS_Item_ID field is equivalent to specifying the containing MCLB's HPSID.

For list boxes and MCLBs with the Selection Type property set to SINGLE or EXTENDED, the newly selected rows or cells replace any existing selection. For Selection Type of MULTIPLE, the new selection is added to any existing selection, thereby allowing non-contiguous selection with multiple calls to the component.

This component causes the identified fields to be selected as if you had clicked on them with the mouse. The selected fields appear in reverse video. You can then use the Get_Selected_Field component to obtain the name and occurrence number of the fields and process them.

HPS_SET_SELECTED_FIELD summarizes how occurrences are selected according to the Selection Type of the list box or MCLB, field occurrence values, and the USE_OLD_SET_SELECTED_FIELD setting in the [AE Runtime] section of the Hps.ini file. In order for the selection to be valid under the default setting of the HPS_SET_SELECTED_FIELD component, that is USE_OLD_SET_SELECTED_FIELD=FALSE, the following conditions must be met:

- HPS_Field_Occur_Start must be greater than zero but less than or equal to database/virtual size.
- HPS_Field_Occur_End must be zero or greater than or equal to start, but less than or equal to database/virtual size.
- For *Single* selection type, HPS_Field_Occur_End must equal HPS_Field_Occur_Start unless HPS_Field_Occur_End is zero.
- If HPS_Field_Occur_End is zero, then for any selection type, the single row specified by HPS_Field_Occur_Start is selected, thus changing the selection type does not affect the rule behavior in this respect.

For list boxes and MCLBs with the Selection Type property set to MULTIPLE or EXTENDED, a range of rows or cells within a column may be selected.

A warning message, Cmp239w, may be displayed to report range specification errors by specifying "Warnings" for the MESSAGES setting in the [AE Runtime] section of the Hps.ini file. Refer to the *INI Settings Reference Guide* for more information about the ini settings. Refer to the *Messages Reference Guide* for more information about warning and error messages.

HPS_SET_SELECTED_FIELD

Selection Type	**Start**	**End**	**Occurrences Selected**			
			C Client with USE_OLD_...=TRUE	**Java Client**	**C Client with USE_OLD_...=FALSE**	**C# Client**
Single	0	0	1	-	-	-
	0	> Start	1	-	-	-
	n	0	n	n	n	n
	n	< Start	n	n	-	n
	n	= Start	n	n	n	n
	n	> Start	n	n	-	n
	n	> maximum	n	-	-	-
	> maximum	0	maximum	-	-	-
	> maximum	< Start	maximum	-	-	-
	> maximum	= Start	maximum	-	-	-
	> maximum	> Start	maximum	-	-	-
Multiple	0	0	-	-	-	-

	0	> Start	1 to End	1 to End	-	1 to end
	n	0	-	-	n	-
	n	< Start	-	-	-	-
	n	= Start	n	n	n	n
	n	> Start	n to End	n to End	n to End	n to end
	n	> maximum	n to maximum	n to maximum	-	n to maximum
	> maximum	0	-	-	-	-
	> maximum	< Start	-	-	-	-
	> maximum	= Start	-	-	-	-
	> maximum	> Start	-	-	-	-
Extended	0	0	-	-	-	-
	0	> Start	End	1 to End	-	1 to end
	n	0	-	-	n	-
	n	< Start	-	-	-	-
	n	= Start	n	n	n	n
	n	> Start	End	n to End	n to End	n to end
	n	> maximum	maximum	n to maximum	-	n to maximum
	> maximum	0	-	-	-	-
	> maximum	< Start	-	-	-	-
	> maximum	= Start	-	-	-	-
	> maximum	> Start	-	-	-	-

USE_OLD_... in the heading refers to the USE_OLD_SET_SELECTED_FIELD setting in the Hps.ini file.
n = integer value greater than 0 but less than or equal to the range set for the list box or MCLB
maximum = current virtual limit of the list box or MCLB
'-' indicates that no occurrence is selected, and the previously selected occurrences, if any, remain as selected.

The occurrences selection for Java client may change in the future release. Selection for Java client does not depend on the USE_OLD_SET_SELECTED_FIELD setting.

Example: HPS_Set_Selected_Field

This example selects rows 1 through 5 of the list box with the system identifier of LISTBOX.

```
map 'LISTBOX' to HPS_ITEM_ID of HPS_SET_SELECTED_FIELD_I
map 1 to HPS_FIELD_OCCUR_START of HPS_SET_SELECTED_FIELD_I
map 5 to HPS_FIELD_OCCUR_END of HPS_SET_SELECTED_FIELD_I
use component HPS_SET_SELECTED_FIELD
```

HPS_Set_Window_Attributes

This section presents the purpose, the syntax, the usage, and how the HPS_Set_Window_Attribute component is supported. See Example: HPS_Set_Window_Attributes for an usage example.

Purpose

Use this component to determine whether a panel is visible or invisible. As is the case with all deferred components, the return code from the component is *always* 1 (Success).
In C, this is a deferred component .

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Set_Window_Attributes component.

HPS_Set_Window_Attributes component syntax

Name	`HPS_Set_Window_Attributes`
System Identifier	`CWINATR`
Input View Name	`HPS_Set_Window_Attributes_I`
Input View Field	`Window_Long_Name character(30)` The window name of the panel.
Input View Field	`HPS_Window_Mode integer(15)` The window display style being set by this component. 0 = invisible, 1 = visible
Output View Name	`HPS_Set_Window_Attributes_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

Remember these restrictions:

- If an invalid window name is passed to the component, no error message is displayed.
- If no window name is given, an error window is displayed.
- If an invalid value is entered into HPS_Window_Mode, an error window is displayed.

The error messages listed above can be suppressed by not setting the value MESSAGES=Errors in the AE Runtime section of the system initialization file (hps.ini).

Example: HPS_Set_Window_Attributes

This example makes a window invisible.

```
map 'WINDOW NAME' to WINDOW_LONG_NAME of HPS_SET_WINDOW_ATTRIBUTES_I
map 1 to HPS_WINDOW_MODE of HPS_SET_WINDOW_ATTRIBUTES_I
use component HPS_SET_WINDOW_ATTRIBUTES
```

HPS_Tbl_Get_Data_State

This section presents the purpose, the syntax, the usage, and how the HPS_Tbl_Get_Data_State component is supported. See Example: HPS_Tbl_Get_Data_State for an usage example.

Purpose

Use this component, with HPS_Tbl_Init_Size, to implement smooth scrolling for a multicolumn list box (MCLB). Smooth scrolling allows you to fetch data from a database in small increments to populate a MCLB view each time a user scrolls the MCLB beyond the bounds of its view---rather than fetching at once all the data the MCLB might ever contain.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Tbl_Get_Data_State component.

HPS_Tbl_Get_Data_State component syntax

Name	`HPS_Tbl_Get_Data_State`
System Identifier	`CTBLDAT`
Input View Name	`HPS_Tbl_Get_Data_State_I`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the multicolumn list box
Output View Name	`HPS_Tbl_Get_Data_State_O`
Output View Field	`Elevator_Pos integer(15)` Virtual occurrence number of the first visible row
Output View Field	`HPS_Virtual_Start_Pos integer(15)` First occurrence of the data source (usually a database) from which data should be mapped to the MCLB view
Output View Field	`HPS_Virtual_End_Pos integer(15)` Last occurrence of the data source (usually a database) from which data should be mapped to the MCLB view
Output View Field	`HPS_View_Start_Pos integer(15)` First occurrence of the MCLB view to which data should be mapped
Output View Field	`HPS_View_End_Pos integer(15)` Last occurrence of the MCLB view to which data should be mapped
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Usage

Whenever an end user scrolls beyond the occurrences in the MCLB view, the system generates an HPS_LB_OUTRANGE event which you can test for. When an HPS_LB_OUTRANGE event occurs, you can use this component to determine which instances to fetch from a database and which occurrences to map to in the MCLB view.

Event Information

When a user scrolls outside an MCLB view, the system generates a system event and returns the following information in the pre-defined system view HPS_EVENT_VIEW:

Information returned in the pre-defined system view HPS_EVENT_VIEW

Field	Value
EVENT_NAME	'HPS_LB_OUTRANGE'

EVENT_QUALIFIER	'UP' (if the user was scrolling up) 'DOWN' (if the user was scrolling down)

This component works only with multicolumn list boxes. If you want to implement smooth scrolling for a single-column list box, you can use a multicolumn list box instead with a single, non-editable column.

Example: HPS_Tbl_Get_Data_State

This example uses the entities summarized in this table, where the Rules Language source code for the rules are given below.

Descriptions of entities

Entity Name	**Entity Type**	**Description**
SMOOTH	Rule	Converses window SMOOTH_WIN and invokes rule SMOOTH_FETCH
SMOOTH_WIN	Window	Contains the MCLB, SMOOTH_MCLB
SMOOTH_WIN_VIEW	View	View for window, SMOOTH_WIN
SMOOTH_MCLB	MCLB	MCLB in window SMOOTH_WINDOW
SMOOTH_MCLB_VIEW	MCLB	View for MCLB, SMOOTH_MCLB
SMOOTH_FETCH	Rule	Fetches data from database for MCLB
SMOOTH_FETCH_IO	View	Input/Output view for SMOOTH_FETCH

RULE SMOOTH

```
*> initialize component with the system identifier of the MCLB <*
map 'SMOOTH_MCLB' to HPS_ITEM_ID of HPS_TBL_INIT_SIZE_I
map 1000 to VIRTUAL_SIZE of HPS_TBL_INIT_SIZE_I
use component HPS_TBL_INIT_SIZE

do
   CONVERSE WINDOW SMOOTH_WIN NOWAIT
while EVENT_SOURCE of HPS_EVENT_VIEW <> 'Exit'
caseof EVENT_SOURCE of HPS_EVENT_VIEW
   case 'SMOOTH MCLB'
      if EVENT_NAME = 'HPS_LB_OUTRANGE'
      *> map the MCLB ID and view into the fetch rule's input view<*
         map 'SMOOTH_MCLB' to HPS_ITEM_ID of SMOOTH_FETCH_IO
         map MCLB_VIEW of SMOOTH_WIN_VIEW to MCLB_VIEW
                       of SMOOTH_FETCH_IO

         *> fetch some data <*
         use rule SMOOTH_FETCH

         *> map the data back into the MCLB view <*
         map MCLB_VIEW of SMOOTH_FETCH_IO to MCLB_VIEW of SMOOTH_WIN_VIEW
      endif
   endcase
enddo
```

.6 **RULE SMOOTH_FETCH**

```
*> initialize component with the system identifier of the MCLB <*
map HPS_ITEM_ID of SMOOTH_FETCH_IO to HPS_ITEM_ID of HPS_TBL_GET_DATA_STATE_I
use component HPS_TBL_GET_DATA_STATE

\*> Use information returned by HPS_TBL_GET_DATA_STATE
    to fetch data from database and map to MCLB view<\*

\*> Fetch data from database starting at occurrence VIRTUAL_START_POS
    of HPS_TBL_GET_DATA_STATE_O and ending at VIRTUAL_END_POS
    of HPS_TBL_GET_DATA_STATE_O <\*
       (insert SQL statements)

\*> Map the data to MCLB_VIEW of SMOOTH_FETCH_IO,
    starting at position VIEW_START_POS of HPS_TBL_GET_DATA_STATE_O
    and ending at VIEW_END_POS of HPS_TBL_GET_DATA_STATE_O <\*
       (insert map statements)
```

HPS_Tbl_Init_Size

This section presents the purpose, the syntax, the usage, and how the HPS_Tbl_Init_Size component is supported. See Example: HPS_Tbl_Init_Size for an usage example.

Purpose

Use this component to inform the system of the virtual size of a multicolumn list box (MCLB). The virtual size is the maximum number of occurrences that can (with scrolling) be displayed in the MCLB---usually the number of occurrences that can be retrieved from a database. The system uses this information to correctly position the elevator within the MCLB scroll bar.
This component is intended to be called only once per set of data. Reset the MCLB state at the time of the component call.
In C, this is a *deferred component* .

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of HPS_Tbl_Init_Size component.

HPS_Tbl_Init_Size component syntax

Name	`HPS_Tbl_Init_Size`
System Identifier	`CTBLSIZ`
Input View Name	`HPS_Tbl_Init_Size_I`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the MCLB
Input View Field	`Virtual_Size integer(15)` Virtual size of the MCLB (the maximum number of occurrences that can, with scrolling, be displayed in the MCLB)
Input View Field	`HPS_Back_Buffer integer(15)` This is not used.

Output View Name	`HPS_Tbl_Init_Size_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

This component works only with multicolumn list boxes. If you want to implement smooth scrolling for a single-column list box, you can use a multicolumn list box instead with a single, non-editable column.
You use this component when the virtual size is so large that it is impractical to make the MCLB view large enough to contain all the occurrences (and to do a single fetch to populate the view before displaying the MCLB). By using this component in conjunction with *HPS_Tbl_Get_Data_State* , you can make the view smaller than the number of occurrences that can potentially be displayed in the MCLB, and then do additional fetches whenever the user scrolls beyond the bounds of the view.
With HPS_TBL_INIT_SIZE, you can only specify the overall data range (plus the back buffer amount, but that's a minor detail).
Runtime then prompts you, by generating HPS_LB_OUTRANGE events, to provide the data it requires for the current physical range. You are required to call HPS_TBL_GET_DATA_STATE to determine the data that runtime wants and you can not override this - you MUST populate the MCLB view with the range of data that runtime tells you.
When you call HPS_TBL_INIT_SIZE, runtime resets the scroll position to row 1 and asks you data from row 1 (this is existing behavior and not related to my fix). This is the only time that the rules code can affect the range of data that should be contained within the view. All subsequent changes to the range of data that should be in the view, notified to you be the HPS_LB_OUTRANGE events, are solely driven by the user and their attempts to scroll outside the current range in the view - runtime will calculate the new range of data required and you MUST honour that request.
Thus the basis for my saying that HPS_TBL_INIT_SIZE should be called only once for any given set of data.
Compare with SET_VIRTUAL_LISTBOX_SIZE in which the application completely drives the subset of data currently in the view. In response of HPS_LB_OUTRANGE events generated when the user attempts to scroll outside the current range in the view, the application is completely responsible for determining the new range of data that should be populated in the view and MUST then tell runtime how the new view contents map to the full range of data by setting the ELEVATOR_POSITION field appropriately.
Thus SET_VIRTUAL_LISTBOX_SIZE is intended to be called multiple times for any given set of data.

Virtual Listbox Size

There is an important distinction between the size of an MCLB, the size of its view, and the virtual size of the MCLB.

- The MCLB contains the instances of the occurring field presented to the end user. For example, an MCLB can contain 10 occurrences that are displayed when a window is conversed.
- The MCLB view contains the instances of the occurring field that can be presented to the end user without an HPS_LB_OUTRANGE condition being generated.
  For example, the MCLB view for an MCLB containing 10 occurrences can itself contain 100 occurrences. As long as the end user does not scroll past these 100 occurrences, the system handles the scrolling and no HPS_LB_OUTRANGE condition is returned.
  If the MCLB view is large enough to accommodate all the instances that can potentially be displayed in the MCLB, then the system handles scrolling without your having to do anything in Rules Language code beyond initially mapping the instances to the MCLB view.
- The virtual size is the maximum number of instances of the occurring field that are potentially available to the end user. For example, if an MCLB contains 10 occurrences and its view 100 occurrences, the virtual size might be 1000 occurrences if there are 1000 occurrences that might be fetched for display in the MCLB. Whenever an end user scrolls beyond the 100 occurrences in the view, the system generates an HPS_LB_OUTRANGE event which you can test for. When an HPS_LB_OUTRANGE event occurs, you can use HPS_Tbl_Get_Data_State to determine which instances to fetch from a database and which occurrences in the MCLB view to map to.

Event Information

When this component is used, the out range event is not generated if the number of rows being set falls within the physical size of the MCLB. If the virtual size being set is greater than the physical size, a system event is generated and the HPS_EVENT_VIEW is populated as follows:

HPS_EVENT_VIEW populated

Field	Value
EVENT_NAME	'HPS_LB_OUTRANGE'
EVENT_QUALIFIER	'INIT'

Example: HPS_Tbl_Init_Size

See the examples under HPS_Tbl_Get_Data_State.

Restore_Altered_Fields

This section presents the purpose, the syntax, the usage, and how the Restored_Altered_Fields component is supported. See Example: Restore_Altered_Fields for an usage example.

Purpose

Use this component to clear altered field flags. An iterative call to Get_Altered_Field loops through altered fields unless you clear the altered field flags by using this component. Use of this component does not change the value in a field; it just resets its status to unaltered.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Restore_Altered_Fields component.

Restore_Altered_Fields component syntax

Name	`Restore_Altered_Fields`
System Identifier	`CCLRALT`
Input View Name	`Restore_Altered_Fields_I`
Input View Field	`Window_Long_Name character(30)`
Output View Name	`Restore_Altered_Fields_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: Restore_Altered_Fields

This example specifies SELECT_CUSTOMER as the window on which to restore altered fields.

```
map 'SELECT_CUSTOMER' to WINDOW_LONG_NAME of RESTORE_ALTERED FIELDS_I
use component RESTORE_ALTERED_FIELDS
```

Set_CloseDown_Notification

This section presents the purpose, the syntax, the usage, and how the Set_CloseDown_Notification component is supported. See Examples: Set_CloseDown_Notification for an usage example.

Purpose

Use this component to set the value of a return code, allowing the application to react to close-down conditions, and enables the *Close* choice on the system menu if it has been disabled.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients. For the Window_Long_Name input view field, the value of asterisk (*) is not supported in Java.

Syntax

Here is the syntax of Set_CloseDown_Notification component.

Set_CloseDown_Notification component syntax

Name	`Set_CloseDown_Notification`

System Identifier	`CAPPOFF`
Input View Name	`Set_CloseDown_Notification_I`
Input View Field	`Window_Long_Name character(30)` Can be set to asterisk (*) to create a global Return_Code for all windows in the application.
Input View Field	`Notify_Return_Code character(50)`
Output View Name	`Set_CloseDown_Notification_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Usage

If the user closes the application by double-clicking the window's system menu symbol, the rule receives the Notify_Return_Code specified in this component. The information contained in the Notify_Return_Code provides the application with an opportunity to react to the close-down condition.
This information appears either in the WINDOW_RETCODE field or the EVENT_SOURCE field of the pre-defined system view HPS_EVENT_VIEW, depending on which mechanism the application is using. If nothing is provided in the Notify_Return_Code field, the *Close* choice is disabled.
Using an asterisk (*) in the Window_Long_Name field to create a global Return_Code affects only windows that are conversed subsequently or that are currently visible. It does not affect windows that are open but invisible (for example a window that is invisible because it was conversed by a rule that uses another rule that has conversed a window that is currently visible).

Examples: Set_CloseDown_Notification

In this example, the Exit is the value of the Close text property for the window.

```
map 'Customer Details' to WINDOW_LONG_NAME of SET_CLOSEDOWN_NOTIFICATION_I
```

In this example, the *Close* system menu choice for the next window conversed is enabled.

```
map 'Exit' to NOTIFY_RETURN_CODE of SET_CLOSEDOWN_NOTIFICATION_I
use component SET_CLOSEDOWN_NOTIFICATION
```

Set_Control_Color_By_ID

This section presents the purpose, the syntax, the usage, and how the Set_Control_Color_By_ID component is supported. See Example: Set_Control_Color_By_ID for an usage example.

Purpose

Use this component to dynamically modify the color property of certain objects. You can use this component to set the color of the background, text, or border of an edit field, multiline edit field, protected field, push button, radio button, check box, list box, group box, combo box, static text, or multicolumn list box (whole box only, not individual cells).
In C, this is a deferred component .

⚠ Because of a Windows restriction, in C you cannot modify the border, text, or background color of push buttons or the border color of other objects in applications to be executed in Microsoft Windows.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients. For a thin client, there is no way to set a separate color for a single MCLB column only for the whole MCLB. For the Field_Attr input view field, the reset value 12 is not supported.

In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for information on other supported components.
For 3270 Converse, setting the background color causes the field to be displayed in reverse-video (black text on a colored background).

- 2 = Border
- 10 = Reset background to original color
- 11 = Reset text to original color
- 12 = Reset border to original color

3270 Converse supports modification of an individual cell in a list box. The input view of a 3270 Converse component contains an INTEGER 15 field for specifying the individual cell. Specify an occurrence number of 0 (zero) to modify the entire list box or column.

Syntax

Here is the syntax of Set_Control_Color_By_ID component.

Set_Control_Color_By_ID component syntax

Name	`Set_Control_Color_By_ID`
System Identifier	`CCTLCID`
Input View Name	`Set_Control_Color_By_ID_I`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the object
Input View Field	`Field_Attr integer(15)` Property to be colored. This field is attached to the system set WINDOW_OBJECT_ATTRIBUTES (SATTR), which has the following values: 0 = Background 1 = Text 2 = Border 10 = Resets background to color originally set in Window Painter 11 = Resets text to color originally set in Window Painter 12 = Resets border to color originally set in Window Painter The value in the Attr_Color field is ignored when you reset the color.
Input View Field	`Attr_Color integer(15)` Color to make the selected property: 0 = White, 1 = Yellow, 2 = Blue 3 = Red, 4 = Not supported, 5 = Not supported 6 = Green, 7 = Magenta, 8 = Black 9 = Aqua This field is attached to the system set WINDOW_OBJECT_COLORS (SCOLORS).
Output View Name	`Set_Control_Color_By_ID_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Set_Control_Color_By_ID

In this example, the background color of the object with the system identifier of Exit is set to red in the next window conversed.

```
map 'Exit' to HPS_ITEM_ID of SET_CONTROL_COLOR_BY_ID_I
map 0 to FIELD_ATTR of SET_CONTROL_COLOR_BY_ID_I
map 3 to ATTR_COLOR of SET_CONTROL_COLOR_BY_ID_I
use component SET_CONTROL_COLOR_BY_ID
```

Set_Control_Mode_By_ID

This section presents the purpose, the syntax, the usage, and how the Set_Control_Mode_By_ID component is supported. See Example: Set_Control_Mode_By_ID for an usage example.

Purpose

Use this component to set the mode of certain objects. An object can be in one of three modes: invisible (not displayed), disabled (displayed but not changeable), or enabled (displayed and changeable). You can use this component to change the mode of an edit field, protected field, static text, combo box, check box, list box, chart, radio button, push button, ellipse, rectangle, or bitmap. You can use this component to disable a hot spot. You can also enable or disable individual columns in a multicolumn list box if HPS_Listbox_Cell_ID is specified.
In C, this is a *deferred component* .

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients; however, the value 3 for the Field_Mode input view field is not supported in Java, and for columns of a multicolumn listbox (MCLB), the value 0 is not supported for the Field_Mode input view field.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.
3270 Converse supports modification of an individual cell in a list box. The input view of a 3270 Converse component contains an INTEGER 15 field for specifying the individual cell. Specify an occurrence number of 0 (zero) to modify the entire list box or column.

Syntax

Here is the syntax of Set_Control_Mode_By_ID component.

Set_Control_Mode_By_ID component syntax

Name	`Set_Control_Mode_By_ID`
System Identifier	`CCTLMID`
Input View Name	`Set_Control_Mode_By_ID_I`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the object
Input View Field	`HPS_Listbox_Cell_ID character(50)` The system identifier (HPSID) of multicolumn list box cell
Input View Field	`Field_Mode integer(15)` This field is attached to the system set WINDOW_OBJECT_MODES (SMODE), which has the following values: 0 = Invisible, disabled 1 = Visible, disabled 2 = Visible, enabled 3 = For mainframe: Invisible, enabled (3270 Converse only) 3= For C: Visible, read-only
Output View Name	`Set_Control_Mode_By_ID_O`

Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

For C Runtime, if a field was created as protected in Window Painter it can be changed to editable at runtime unless the ALLOW_CHANGE_PROTECTED_FIELDS ini setting is changed to FALSE. This ini setting defaults to TRUE.

Example: Set_Control_Mode_By_ID

In this example, the object with the system identifier of Save is disabled in the next window conversed.

```
map 'Save' to HPS_ITEM_ID of SET_CONTROL_MODE_BY_ID_I
map 1 to FIELD_MODE of SET_CONTROL_MODE_BY_ID_I
use component SET_CONTROL_MODE_BY_ID
```

Set_Control_RGBColor_By_ID

This section presents the purpose, the syntax, the usage, and how the Set_Control_RGBColor_By_ID component is supported. See Example: Set_Control_RGBColor_By_ID for an usage example.

Purpose

Use this component to dynamically modify the color property of certain objects. You can use this component to set the color of the background, text, or border of an edit field, multiline edit field, protected field, push button, radio button, check box, list box, group box, combo box, static text, or multicolumn list box (whole box only, not individual cells).

Note
Because of a Windows restriction, you cannot modify the border, text, or background color of push buttons or the border color of other objects in applications to be executed in Microsoft Windows.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients. For a thin client, there is no way to set a separate color for a single MCLB column only for the whole MCLB. For the Field_Attr input view field, the reset value 12 is not supported.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Set_Control_RGBColor_By_ID component.

Set_Control_RGBColor_By_ID component syntax

Name	`Set_Control_RGBColor_By_ID`
System Identifier	`CSCRGBC`
Input View Name	`Set_Control_RGBColor_By_ID_I`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the object

Input View Field	`Field_Attr integer(15)` Property to be colored. This field is attached to the system set WINDOW_OBJECT_ATTRIBUTES, which has the following values: 0 = Background 1 = Text 2 = Border 10 = Resets background to color originally set in Window Painter 11 = Resets text to color originally set in Window Painter 12 = Resets border to color originally set in Window Painter The values in the following color fields are ignored when you reset the color.
Input View Field	`Red_Color integer(15)` Value for the red component of the color (0--255).
Input View Field	`Green_Color integer(15)` Value for the green component of the color (0--255).
Input View Field	`Blue_Color integer(15)` Value for the blue component of the color (0--255).
Output View Name	`Set_Control_RGBColor_By_ID_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: Set_Control_RGBColor_By_ID

This example sets to red the background of the object with the system identifier of Exit.

```
map 'Exit' to HPS_ITEM_ID of SET_CONTROL_RGBCOLOR_BY_ID_I
map 0 to FIELD_ATTR of SET_CONTROL_RGBCOLOR_BY_ID_I
map 255 to RED_COLOR of SET_CONTROL_RGBCOLOR_BY_ID_I
map 0 to GREEN_COLOR of SET_CONTROL_RGBCOLOR_BY_ID_I
map 0 to BLUE_COLOR of SET_CONTROL_RGBCOLOR_BY_ID_I
use component SET_CONTROL_RGBCOLOR_BY_ID
```

Set_Cursor_Field

This section presents the purpose, the syntax, the usage, and how the Set_Cursor_Field component is supported. See Example: Set_Cursor_Field for an usage example.

Purpose

Use this component to determine the field in which the cursor appears when the window is reconversed. This allows you to override the default cursor positioning. A field selected with this component is highlighted. Thus, a user can immediately begin to enter text, overwriting any text already in the field.
A field name and a view name must be provided and must match a field on the currently conversed window.
Specify an occurrence number of zero if the field is not in a list box or if you want to position the cursor on the first occurrence of the field.

⚠ This component does not support list boxes in an IBM AIX environment. You must therefore specify zero as the occurrence number for applications running in AIX.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported

Interface Components for more information. 3270 Converse does not highlight the field selected with this component.

Syntax

Here is the syntax of Set_Cursor_Field component.

Set_Cursor_Field component syntax

Name	`Set_Cursor_Field`
System Identifier	`CCURPOS`
Input View Name	`Set_Cursor_Field_I`
Input View Field	`Field_Long_Name character(30)` The name of the field in the hierarchy diagram to which the field in the window is linked
Input View Field	`View_Long_Name character(30)` The view that includes the field mapped to FIELD_LONG_NAME
Input View Field	`Listbox_Occur integer(15)` Occurrence number of field within a list box (ignored if not a list box field)
Output View Name	`Set_Cursor_Field_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: Set_Cursor_Field

This example sets the cursor on the field FIELD_1.

```
map 'FIELD_1' to FIELD_LONG_NAME of SET_CURSOR_FIELD_I
map 'VIEW_1' to VIEW_LONG_NAME of SET_CURSOR_FIELD_I
use component of SET_CURSOR_FIELD
```

Set_Default_Push_Button

This section presents the purpose, the syntax, the usage, and how the Set_Default_Push_Button component is supported. See Example: Set_Default_Push_Button for an usage example.

Purpose

Use this component to set the button with a specified system identifier as the default push button.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Set_Default_Push_Button component.

Set_Default_Push_Button component

Name	`Set_Default_Push_Button`
System Identifier	`CDEFBUT`
Input View Name	`Set_Default_Push_Button_I`
Input View Field	`Pushbutton_ID character(50)`

Output View Name	`Set_Default_Push_Button_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: Set_Default_Push_Button

This example sets the button to the button with the system identifier of Pushbutton_ID.

```
map 'SEARCH' to PUSHBUTTON_ID of SET_DEFAULT_PUSH_BUTTON_I
use component SET_DEFAULT_PUSH_BUTTON
```

Set_Field_Blink_By_ID

This section presents the purpose, the syntax, the usage, and how the Set_Field_Blink_By_ID component is supported. See Examples: Set_Field_Blink_By_ID for an usage example.

Purpose

Use this component to dynamically modify a highlighting property of a field. You can use this component to set an edit field, protected field or list box to be displayed as blinking in the current color or to return it to normal non-blinking. Returning to normal non-blinking may also be accomplished by using the Set_Field_Color component or the Clear_Window_Changes component.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.
Blinking fields only display on 3270 devices that support extended data streams; on other 3270 devices, the change is accepted but the data is not displayed as blinking.

This component is only valid in the 3270 Converse environment.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Set_Field_Blink_By_ID component.

Set_Field_Blink_By_ID component syntax

Name	`Set_Field_Blink_By_ID`
System Identifier	`CFLDBLK`
Input View Name	`Set_Field_Blink_By_ID_I`
Input View Field	`Window_Long_Name character(30)` The name of the window that contains the field to be modified. This field must be set to the current window name, i.e. the window conversed by the calling rule.
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the object.
Input View Field	`HPS_Listbox_Cell_ID character(50)` The system identifier (HPSID) of the multicolumn list box cell.

Input View Field	`Field_Occur integer(15)` The occurrence number of the record in the list box to be affected, if the item is a list box. Specify an occurrence number of zero to modify all occurrences.
Input View Field	`Field_Blink_State integer(15)` Sets the item to blinking or normal. 0 = Normal, 1 = Blinking.
Output View Name	`Set_Field_Blink_By_ID_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Examples: Set_Field_Blink_By_ID

For an edit field, which has a system identifier 'CUSTOMER_LAST_NAME', to be blinking, the component should be used in the following way.

```
map 'CUSTOMER_LAST_NAME' to HPS_ITEM_ID of SET_FIELD_BLINK_BY_ID_I
map 'CUSTOMER_DTL_DIS' to WINDOW_LONG_NAME of SET_FIELD_BLINK_BY_ID_I
map 1 to FIELD_BLINK_STATE of SET_FIELD_BLINK_BY_ID_I
use component set_field_blink_by_id
```

For an MCLB, this component is used to make the 5th row of the Customer_Last_Name_Col column to blink.

```
map 'CUSTOMER_LAST_NAME_COLL' to HPS_LISTBOX_CELL_ID of SET_FIELD_BLINK_BY_ID_I
map 'CUSTOMER_LST_DIS' to WINDOW_LONG_NAME of SET_FIELD_BLINK_BY_ID_I
map 5 to FIELD_OCCUR of SET_FIELD_BLINK_BY_ID_I
map 1 to FIELD_BLINK_STATE of SET_FIELD_BLINK_BY_ID_I
use component set_field_blink_by_id
```

Set_Field_Color

This section presents the purpose, the syntax, the usage, and how the Set_Field_Color component is supported. See Examples: Set_Field_Color for an usage example.

Purpose

Use this component to dynamically modify a color property of a field. You can use this component to set the color of the background, text, or border of an edit field, protected field, check box, radio button, or list box. If no field name is given, this component sets all fields for the given view. You can also change the color of a whole multicolumn list box, but not individual cells within it.
In C, this is a deferred component.

⚠ Because of a Windows restriction, you cannot change the border, text, or background color of push buttons or the border color of other objects in applications to be executed in Microsoft Windows.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients. For the Field_Attr input view field, the reset value 12 is not supported.
In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for information on other components.
For 3270 Converse applications executing on the mainframe, this component lets you modify text color only; any value specified in Field_Attr is

ignored. Colors displayed on the 3270 terminal device do not map exactly to colors displayed on the workstation. The component has no effect on 3270 terminal devices that do not support color. Both a field and a view must be specified; otherwise the Return_Code is set to failure.
For 3270 Converse applications, you can use the keywords *EDIT, *TEXT and *ALL in the Field_Long_Name field of the input view. *EDIT modifies all edit fields within the specified view. *TEXT modifies all the output text fields within the specified view, except text literal fields. *ALL modifies all fields within the specified view, except text literal fields. You can use *ALL in conjunction with a nonzero occurrence number to change an entire row in a list box. The occurrence numbers that can be used are:

- 2 = Border
- 10 = Reset background to original color
- 11 = Reset text to original color
- 12 = Reset border to original color

Syntax

Here is the syntax of Set_Field_Color component.

Set_Field_Color component syntax

Name	`Set_Field_Color`
System Identifier	`CFLDCOL`
Input View Name	`Set_Field_Color_I`
Input View Field	`View_Long_Name character(30)` The view that includes the field mapped to Field_Long_Name. You must enter a value in this field when changing the color of a multicolumn list box. This field is optional for other objects.
Input View Field	`Field_Long_Name character(30)` The name of the field in the hierarchy diagram to which the field in the window is linked
Input View Field	`Field_Attr integer(15)` Property to be colored. This field is attached to the system set WINDOW_OBJECT_ATTRIBUTES (SATTR), which has the following values: • 0 = Background, 1 = Text, 2 = Border • 10 = Resets background to color originally set in Window Painter • 11 = Resets text to color originally set in Window Painter • 12 = Resets border to color originally set in Window Painter The value in the Attr_Color field is ignored when you reset the color.
Input View Field	`Attr_Color integer(15)` Color to make the selected property. This field is attached to the system set WINDOW_OBJECT_COLORS (SCOLORS), which has the following values: 0 = White, 1 = Yellow, 2 = Blue 3 = Red, 4 = Not supported, 5 = Not supported 6 = Green, 7 = Magenta, 8 = Black 9 = Aqua
Input View Field	`Field_Occur integer(15)` On the mainframe, this is the occurrence number of the record in the list box to be affected. Specify an occurrence number of zero to modify all occurrences. On the workstation, you can change the whole list box, but you cannot change individual occurrences within it. You can ignore this field when working on a workstation.
Output View Name	`Set_Field_Color_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Examples: Set_Field_Color

This example specifies SELECT_CUSTOMER as the view that owns the field whose color is to be set.

map 'SELECT_CUSTOMER' to VIEW_LONG_NAME of SET_FIELD_COLOR_I

use component SET_FIELD_COLOR
This example sets the scope of the component to include all fields in the view SELECT_CUSTOMER.

```
map ' ' to FIELD_LONG_NAME of SET_FIELD_COLOR_I
use component SET_FIELD_COLOR
```

This example specifies that the target of the SET_FIELD_COLOR component is the border of all the fields in the view SELECT_CUSTOMER.

```
map 2 to FIELD_ATTR of SET_FIELD_COLOR_I
use component SET_FIELD_COLOR
```

This example converts to black the border of all the fields in the view SELECT_CUSTOMER.

```
map 8 to ATTR_COLOR of SET_FIELD_COLOR_I
use component SET_FIELD_COLOR
```

Set_Field_Message

This section presents the purpose, the syntax, the usage, and how the Set_Field_Message component is supported. See Examples: Set_Field_Message for an usage example.

Purpose

Use this component to mark an edit field that is in error and displays an error message. When the window is invoked, the background of the field specified turns red and the error message, determined by the component, appears in a window next to the field. This window is modeless and remains visible until the user selects *OK.* A field can display only one message at a time.
In C, this is a deferred component .

⚠ This component applies only to unprotected edit fields. On the Windows workstation, this excludes fields in list boxes. This is a change from earlier versions of the product.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

3270 Converse highlights fields in error with inverse video red on 3270 terminal devices that support color and a high-intensity display on 3270 devices that do not support color. The cursor is placed on the field in error.

The message that the parameters set in the input view specifies is edited with the text arguments and displayed on line 22 of the 3270 device. If it is longer than 79 characters, the message is displayed in a scrollable pop-up message as close to the field as possible without overlaying it. Any message text that cannot be displayed is replaced by periods.

This component can manage up to 32 kilobytes of message text data for all the messages of a set. Because it displays messages longer than 79 characters in a bordered pop-up message, only 72 bytes of message text data per line can be displayed on the 3270 terminal device. Be careful when creating message text because the insertion of arguments can cause the line to be truncated.

Syntax

Here is the syntax of Set_Field_Message component.

Set_Field_Message component syntax

Name	`Set_Field_Message`
System Identifier	`CFLDMSG`
Input View Name	`Set_Field_Message_I`
Input View Field	`Text_Code integer(15)` A number indicating which message is to be used from the error set. This field specifies the error message within the set to be displayed.
Input View Field	`Text_Arg1 character(18)` First text input in error message.
Input View Field	`Text_Arg2 character(18)` Second text input in error message.
Input View Field	`Text_Arg3 character(18)` Third text input in error message.
Input View Field	`Field_Long_Name character(30)` The name of the field in the hierarchy diagram to which the field in the window is linked. This field specifies the field in error. The View_Long_Name field specifies the name of the owning view.
Input View Field	`View_Long_Name character(30)` The view that includes the field mapped to Field_Long_Name.
Input View Field	`Field_Occur integer(15)` Occurrence number. This field specifies an instance of an occurring view.
Input View Field	`Message_Set_Name character(8)` The implementation name of the reference file without the REF extension. This field specifies the implementation name of the set that contains the error message.
Output View Name	`Set_Field_Message_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

You can construct up to three customized inputs by giving values to Text_Arg1, Text_Arg2, and Text_Arg3. These strings are then inserted into the generic error message selected from the error message set. The percent character (%) serves as a place holder in the messages for the Text_Arg inputs. Thus %1 is the place holder for Text_Arg1, %2 for Text_Arg2, and %3 for Text_Arg3. The at sign character (@) is also recognized as a place holder when converting older applications, but should not be used when designing new applications. For example, the message

```
%1, try another automobile type. The one you chose does not exist.
```

can be customized by mapping a field containing the user's name into Text_Arg1. Thus, the user name appears in the "%1" position.

Examples: Set_Field_Message

This example specifies the set with the implementation name ERRMSGS as the set from which to retrieve messages.

```
map 'ERRMSGS' to MESSAGE_SET_NAME of SET_FIELD_MESSAGE_I
use component SET_FIELD_MESSAGE
```

This example specifies that the message corresponding to the value RECORD_NOT_FOUND in the set Error_Messages is to be displayed. Note that this must be the set whose implementation name was mapped to MESSAGE_SET_NAME above.

```
map RECORD_NOT_FOUND in ERROR_MESSAGES to TEXT_CODE of SET_FIELD_MESSAGE_I
use component SET_FIELD_MESSAGE
```

This example specifies RESERVATION_NUMBER as the field on which to set the message.

```
map 'RESERVATION_NUMBER' to FIELD_LONG_NAME of SET_FIELD_MESSAGE_I
use component SET_FIELD_MESSAGE
```

This example specifies RESERVATION as the view on which to set the message.

```
map 'RESERVATION' to VIEW_LONG_NAME of SET_FIELD_MESSAGE_I
use component SET_FIELD_MESSAGE
```

Set_Field_Mode

This section presents the purpose, the syntax, the usage, and how the Set_Field_Mode component is supported. See Examples: Set_Field_Mode for an usage example.

Purpose

Use this component to change the display properties and edit mode of a field. For example, an input field can be disabled for further input or the field can be made invisible. Text fields can be made invisible only; enabling and disabling text fields are irrelevant. Edit fields can be made invisible and also be disabled for input. After edit fields are disabled, they can be re-enabled for input. This change is for the entire object, both the field itself and the data within it.
In C, this is a deferred component .

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.
In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Set_Field_Mode component.

Set_Field_Mode component syntax

Name	`Set_Field_Mode`
System Identifier	`CFLDMOD`

Input View Name	`Set_Field_Mode_I`
Input View Field	`View_Long_Name character(30)` The view that includes the field mapped to Field_Long_Name.
Input View Field	`Field_Long_Name character(30)` The name of the field in the hierarchy diagram to which the field in the window is linked.
Input View Field	`Field_Mode integer(15)` This field is attached to the set WINDOW_OBJECT_MODES (SMODE), which has the following values: 0 = Invisible, disabled 1 = Visible, disabled 2 = Visible, enabled 3 = For mainframe: Invisible, enabled (3270 Converse only) 3= For C: Visible, read-only
Input View Field	`Field_Occur integer(15)` On the mainframe, this is the occurrence number of the record in a list box to be affected. Specify an occurrence number of zero to modify all occurrences. On the workstation, you can change the whole list box but cannot change individual occurrences within it. You can ignore this field when working on a workstation.
Output View Name	`Set_Field_Mode_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

You cannot use this component to make a protected field into an editable one because they are two distinct objects. However, for C Runtime, even if a field was created as protected in Window Painter it can be changed to editable at runtime unless the ALLOW_CHANGE_PROTECTED_FIELDS ini setting is changed to FALSE. This ini setting defaults to TRUE.

If the Field_Long_Name contains the value *TEXT, all output text fields are subject to the mode change, regardless of the contents of VIEW_LONG_NAME. Text literal fields, however, remain unaffected. If the field name parameter contains the value *EDIT, all editable fields and combo boxes are subject to the mode change. By specifying both a view and field name, the mode of any object that has a field name associated with it can be changed. This includes check boxes, radio buttons, multiline edit fields, edit fields, combo boxes, list boxes, and individual columns (but not specific cells) of multicolumn list boxes.

Examples: Set_Field_Mode

This example sets the scope for the component to include all editable fields in the window.

```
map '*EDIT' to FIELD_LONG_NAME of SET_FIELD_MODE_I
use component SET_FIELD_MODE
```

This example sets all editable fields in all views to invisible.

```
map INVISIBLE in WINDOW_OBJECT_MODES to FIELD_MODE OF SET_FIELD_MODE_I
use component SET_FIELD_MODE
```

Set_Field_Numeric_By_ID

This section presents the purpose, the syntax, the usage, and how the Set_Field_Numeric_By_ID component is supported. See Examples: Set_Field_Numeric_By_ID for an usage example.

Purpose

Use this component to dynamically modify the entry property of a field. You can use this component to set an edit field or list box edit field to accept only numeric input or only alphanumeric input. This is a feature of the 3270 device that causes the keyboard to lock if non-numeric characters are entered when in numeric mode.

> ⚠ This component is only valid in the 3270 Converse environment and functions for all 3270 devices that support the numeric lock feature.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Set_Field_Numeric_By_ID component.

Set_Field_Numeric_By_ID component syntax

Name	`Set_Field_Numeric_By_ID`
System Identifier	`CFLDNUM`
Input View Name	`Set_Field_Numeric_By_ID_I`
Input View Field	`Window_Long_Name character(30)` The name of the window that contains the field to be modified. This field must be set to the current window name, that is, the window conversed by the calling rule.
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the object.
Input View Field	`HPS_Listbox_Cell_ID character(50)` The system identifier (HPSID) of the multicolumn list box cell
Input View Field	`Field_Occur integer(15)` The occurrence number of the record in the list box to be affected, if the item is a list box. Specify an occurrence number of zero to modify all occurrences.
Input View Field	`Field_Attribute integer(15)` The property to set for the selected field. 0 = Alphanumeric, 1 = Numeric
Output View Name	`Set_Field_Numeric_By_ID_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Examples: Set_Field_Numeric_By_ID

For an edit field, which has a system identifier CUSTOMER_ID, to be made alphanumeric, the component can be used in the following way.

```
map 'CUSTOMER_ID' to HPS_ITEM_ID of SET_FIELD_NUMERIC_BY_ID_I
map 'CUSTOMER_DTL_DIS' to WINDOW_LONG_NAME of SET_FIELD_NUMERIC_BY_ID_I
map 0 to FIELD_ATTRIBUTE of SET_FIELD_NUMERIC_BY_ID_I
use component SET_FIELD_NUMERIC_BY_ID
```

For an MCLB, this component is used to make the CUSTOMER_ID column numeric.

```
map 'CUSTOMER_ID_COL' to HPS_LISTBOX_CELL_ID of SET_FIELD_NUMERIC_BY_ID_I
map 'CUSTOMER_LST_DIS' to WINDOW_LONG_NAME of SET_FIELD_NUMERIC_BY_ID_I
map 1 to FIELD_ATTRIBUTE of SET_FIELD_NUMERIC_BY_ID_I
use component SET_FIELD_NUMERIC_BY_ID
```

Set_Field_Picture

This section presents the purpose, the syntax, the usage, and how the Set_Field_Picture component is supported. See Example: Set_Field_Picture for an usage example.

Purpose

Use this component to dynamically set the display picture of a field and scale for the next converse, including edit fields, protected fields, list boxes, or multicolumn list boxes (MCLBs). A blank picture string is ignored. This component supports only the modification of numeric picture strings. The date and time picture string can be modified only if the field is declared as a date or time field.
In C, this is a *deferred component*.

Note
This component functions only on a field that already has a display picture defined.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients. The Java client does not support the Scale_Percent property (input view field).
In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information. The date and time picture cannot be modified in a 3270 use of this component.

Syntax

Here is the syntax of Set_Field_Picture component.

Set_Field_Picture component syntax

Name	`Set_Field_Picture`
System Identifier	`CPICSIZ`
Input View Name	`Set_Field_Picture_I`
Input View Field	`View_Long_Name character(30)` The view that includes the field mapped to Field_Long_Name.
Input View Field	`Field_Long_Name character(30)` The name of the field in the hierarchy diagram to which the field in the window is linked.
Input View Field	`Picture_Name character(30)` Picture string, such as SZZZ.99.
Input View Field	`Scale_Percent integer(15)` Set the number of digits to be displayed to the right of the decimal point. This property is not supported in a Java client.

Output View Name	`Set_Field_Picture_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Set_Field_Picture

This example sets the display picture for the numeric field FIELD_1.

```
map 'FIELD_1' to FIELD_LONG_NAME of SET_FIELD_PICTURE_I
map 'VIEW_1' to VIEW_LONG_NAME of SET_FIELD_PICTURE_I
map '99999.99' to PICTURE_NAME of SET_FIELD_PICTURE_I
map 2 to SCALE_PERCENT of SET_FIELD_PICTURE_I
use component of SET_FIELD_PICTURE
```

Set_First_Visible_Occurrence

This section presents the purpose, the syntax, the usage, and how the Set_First_Visible_Occurence component is supported. See Example: Set_First_Visible_Occurrence for an usage example.

Purpose

Use this component to set a specific occurrence of an occurring view that is the first occurrence displayed in the window. For SET_FIRST_VISIBLE_OCCURRENCE, the value of FIELD_OCCUR must be within the virtual range. If the value of FIELD_OCCUR exceeds the maximal/min_value, FIELD_OCCUR is set to maximal/min_value. The maximum/minimum possible value is calculated as follows:

```
max_value = virtual_end - visible_occur + 1
min_value = virtual_start
```

After invoking this component, the specified field becomes the first visible occurrence when the window is reconversed.

In C, this is a deferred component .

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information on other components.

For 3270 Converse applications executing on the mainframe, the Return_Code is set to 0 (Failure) if the specified occurrence number is not presently available or invalid; the component does not use the occurrence number closest to the one specified.

3270 Converse repositions a list box so that the maximum number of occurrences are displayed. For example, if a list box size is 25, and the number of visible occurrences is 12, then if the first visible occurrence is set to 18, occurrences 14 through 25 are displayed.

Syntax

Here is the syntax of Set_First_Visible_Occurrence component.
Set_First_Visible_Occurrence component

Name	`Set_First_Visible_Occurrence`
System Identifier	`CSETFST`
Input View Name	`Set_First_Visible_Occurrence_I`
Input View Field	`View_Long_Name character(30)`
Input View Field	`Field_Occur integer(15)` Occurrence number
Output View Name	`Set_First_Visible_Occurrence_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

The occurrence number must be chosen from the occurrences available in the scrollable area. The range of occurrences in the scrollable area is determined by the elevator position and number of records as set by the Set_Virtual_Listbox_Size component. If an unavailable occurrence number is chosen, the number closest to the specified number is used. You can use this component only on occurring views or multicolumn list boxes (MCLBs).

⚠ If a virtual list box has been defined, the field occurrence used as input must also be a virtual occurrence number.

If both this component and the Set_Last_Visible_Occurrence component are used, the component last executed overrides any prior actions.

Example: Set_First_Visible_Occurrence

This example sets the first visible occurrence:

```
map 'NC_TABLE_SL' to VIEW_LONG_NAME of SET_FIRST_VISIBLE_OCCURRENCE_I
map 4 to FIELD_OCCUR of SET_FIRST_VISIBLE_OCCURRENCE_I
use component SET_FIRST_VISIBLE_OCCURRENCE
```

Set_Help_File_Name

This section presents the purpose, the syntax, the usage, and how the Set_Help_File_Name component is supported. See Example: Set_Help_File_Name for an usage example.

Purpose

Use this component to specify the name of the online help file for an application. Once the help file name is set, the SHOW_HELP_TOPIC component can be used to display help for the required keyword, and context-sensitive help can be invoked depending on the embedded system identifiers in the help file.

In C, this is a deferred component .

The setting `GUI_HELP_DIR=` must be added to the AE Runtime section of the system initialization file (hps.ini).

Use this component to link only the native GUI Help files with the application. Do not use this component if you are using the built-in help features of AppBuilder.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.
In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Set_Help_File_Name component.

Set_Help_File_Name component syntax

Name	`Set_Help_File_Name`
System Identifier	`CSETHLP`
Input View Name	`Set_Help_File_Name_I`
Input View Field	`Help_File_Name character(50)` When this component executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.
Output View Name	`Set_Help_File_Name_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred components, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Set_Help_File_Name

This example specifies AEHLPEX.HLP as the Help file to be used in this application.

```
map 'AEHLPEX.HLP' to HELP_FILE_NAME of SET_HELP_FILE_NAME_I
use component SET_HELP_FILE_NAME
```

Set_Item_Text

This section presents the purpose, the syntax, the usage, and how the Set_Item_Text component is supported. See Example: Set_Item_Text for an usage example.

Purpose

Use this component to change the text of a push button, radio button, group box, check box, static text, or menu choice. Because this component does not change the size of the object, you are responsible for making sure the new text fits.
In C, this is a deferred component.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.
In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Set_Item_Text component.

Set_Item_Text component syntax

Name	`Set_Item_Text`
System Identifier	{CITMTXT}}
Input View Name	`Set_Item_Text_I`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the panel object
Input View Field	`HPS_Item_Text character(50)` New text to be displayed. You can pass a mnemonic indicator, an ampersand (&) for Windows, as part of the text. At execution, the character after the indicator is underlined and functions as a menu choice mnemonic.
Output View Name	`Set_Item_Text_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Set_Item_Text

In this example the text on the menu choice with the system identifier of Pulldown 1 is changed to *Replace* in the next window conversed.

```
map 'Pulldown 1' to HPS_ITEM_ID of SET_ITEM_TEXT_I
map '&Replace' to HPS_ITEM_TEXT of SET_ITEM_TEXT_I
use component SET_ITEM_TEXT
```

Set_Last_Visible_Occurrence

This section presents the purpose, the syntax, the usage, and how the Set_Last_Visible_Occurrence component is supported. See Example: Set_Last_Visible_Occurrence for an usage example.

Purpose

Use this component to set a specific occurrence of an occurring view to be the last occurrence displayed in the window. For SET_LAST_VISIBLE_OCCURRENCE, if the value of FIELD_OCCUR exceeds the max_value/min_value, FIELD_OCCUR is set to max_value/min_value. The maximum/minimum possible value is calculated as follows:

```
max_value = virtual_end
min_value = virtual_start + visible_occur \-1
```

Where virtual_start, virtual_end are the start, end index of current virtual range.

After invoking this component, the specified field becomes the last visible occurrence when the window is reconversed.

In C, this is a deferred component.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for information on other components.

For 3270 Converse applications executing on the mainframe, the Return_Code is set to failure if the specified occurrence number is not presently available or invalid; the component does not use the occurrence number closest to the one specified.

3270 Converse repositions a list box so that the maximum number of occurrences are displayed. For example, if a list box size is 25, and the number of visible occurrences is 12, then if the last visible occurrence is set to 8, occurrences 1through 12 are displayed.

Syntax

Here is the syntax of Set_Last_Visible_Occurrence component.

Set_Last_Visible_Occurrence component syntax

Name	`Set_Last_Visible_Occurrence`
System Identifier	`CSETLST`
Input View Name	`Set_Last_Visible_Occurrence_I`
Input View Field	`View_Long_Name character(30)`
Input View Field	`Field_Occur integer(15)` Occurrence number
Output View Name	`Set_Last_Visible_Occurrence_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

The occurrence number must be chosen from the occurrences available in the scrollable area. The range of occurrences in the scrollable area is determined by the elevator position and number of records as set by the Set_Virtual_Listbox_Size component. If an occurrence number not presently available is chosen, the number closest to that specified is used. You can use this component only on occurring views or multicolumn list boxes (MCLBs).

⚠ If a virtual list box has been defined, the field occurrence used as input must also be a virtual occurrence number.

If both this component and the Set_First_Visible_Occurrence component are used, the component last executed overrides any prior actions.

Example: Set_Last_Visible_Occurrence

This example sets an occurrence.

```
map 'NC_TABLE_SL' to VIEW_LONG_NAME of SET_LAST_VISIBLE_OCCURRENCE_I
map 15 to FIELD_OCCUR of SET_LAST_VISIBLE_OCCURRENCE_I
use component SET_LAST_VISIBLE_OCCURRENCE
```

Set_Menu_Mode

This section presents the purpose, the syntax, the usage, and how the Set_Menu_Mode component is supported. See Example: Set_Menu_Mode for an usage example.

Purpose

Use this component to enable or disable a particular menu choice. When a menu choice is active, its items are displayed in black letters. However, a menu choice displayed with gray letters (dimmed) is inactive, preventing the user from selecting it. This component acts like a toggle, so that a second call reverses the state set by the first call.

In C, this is a deferred component.

For more functionality, you may use Set_Menu_Mode_By_ID. To determine the status (whether a menu choice is enabled or disabled), you may use Get_Menu_Mode_By_ID.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

On color 3270 terminals, enabled menu choices are displayed with white text and disabled menu choices with blue text. On monochrome 3270 terminals, enabled menu choices are highlighted.

Syntax

Here is the syntax of Set_Menu_Mode component.

Set_Menu_Mode component

Name	`Set_Menu_Mode`
System Identifier	`CMNUMOD`
Input View Name	`Set_Menu_Mode_I`
Input View Field	`Menu_Name character(50)` Name of menu
Input View Field	`Pulldown_Name character(30)` Name of menu choice
Output View Name	`Set_Menu_Mode_O`

Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

If the named menu has no pull-down menus, set Menu_Name blank and set Pulldown_Name to the name of the menu choice. This causes the menu to be displayed on a menu bar in the appropriate mode.

⚠ On the workstation, when a window is conversed after displaying a window, all menu choices are re-enabled. You must call this component again to disable a menu choice.

Unlike most other component fields, the Menu_Name and Pulldown_Name are case sensitive. If either of these names contain a mnemonic marker (& or ~) or a function key, do not include it when mapping the name into the appropriate field.

Example: Set_Menu_Mode

In this example, the mode of the *Exit* choice under the *File* menu is toggled. In C, this is toggled in the next window conversed. In Java, this acts on the current window.

```
map 'File' to MENU_NAME of SET_MENU_MODE_I
map 'Exit' to PULLDOWN_NAME of SET_MENU_MODE_I
use component SET_MENU_MODE
```

If the **Exit** menu choice was originally enabled, it is now disabled. If it was disabled, it is now enabled.

Set_Menu_Mode_By_ID

This section presents the purpose, the syntax, the usage, and how the Set_Menu_Mode_By_ID component is supported. See Example: Set_Menu_Mode_By_ID for an usage example.

Purpose

Use this component to set the status of either of the two independent properties of a menu choice: whether the menu choice is enabled or whether the menu choice is checked. With each use of this component you can set the status of one these two properties. The Enabled property refers to whether the menu item is enabled (active and selectable) or disabled (inactive and grayed out). The Checked property refers to whether the menu item is checked (has a check mark next to it) or not (has no check mark next to it).

This component is the "set" version of Get_Menu_Mode_By_ID. If you set Enabled to 1, the item becomes active; if set to 0, it becomes inactive. If you set Checked to 1, a check mark is placed next to the item; if set to 0, any check mark is removed. Only one property can be changed at a time. To change both, use the component twice, using a different value for the HPS_MItem_Attribute field each time.

This component is similar to Set_Menu_Mode but has more functionality; it can set the check mark as well as the enabled status. It is also more efficient because it uses the system identifier (HPSID) of the menu item rather than its name.

In C, this is a deferred components .

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Set_Menu_Mode_By_ID component.

Set_Menu_Mode_By_ID component syntax

Name	`Set_Menu_Mode_By_ID`
System Identifier	`CMNUMID`
Input View Name	`Set_Menu_Mode_By_ID_I`
Input View Field	`HPS_Item_ID character(50)` The system identifier (HPSID) of the menu item
Input View Field	`HPS_MItem_Attribute integer(15)` The property of the menu choice you want to set: • 0 = Enabled - whether enabled (active) or disabled (grayed out) • 1 = Checked - whether it has a check mark next to it or not This field is attached to the system set MENU_ITEM_ATTRIBUTES (SMIATTR). (This is not supported by 3270 Converse.)
Input View Field	`HPS_MItem_State integer(15)` The status of the property specified on the input view. (Only one property can be set per call of this component.) • 0 = For Enabled, this means the menu choice is disabled (grayed out); For Checked this means the menu choice is not checked. • 1 = For Enabled, this means the menu choice is enabled (active); For Checked this means the menu choice has a check mark next to it. This field is attached to the system set MENU_ITEM_STATE (SMIST).
Output View Name	`Set_Menu_Mode_By_ID_O`
Output View Field	`{Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Set_Menu_Mode_By_ID

In this example, the *Save* choice in the menu is disabled. For C, this is in the next window conversed. This does not change whether the menu choice is checked or not.

```
map 'Save' to HPS_ITEM_ID of SET_MENU_MODE_BY_ID_I
map 0 to HPS_MITEM_ATTRIBUTE of SET_MENU_MODE_BY_ID_I
map 0 to HPS_MITEM_STATE of SET_MENU_MODE_BY_ID_I
use component SET_MENU_MODE_BY_ID
```

The first map statement selects the menu choice (Save). The second map statement selects the Enable property. The third map statement sets that Enable property to off or disabled, which disables the menu choice.

Set_PopUp_Position

This section presents the purpose, the syntax, the usage, and how the Set_PopUp_Position component is supported. See Example: Set_Popup_Position for an usage example.

Purpose

Use this component to change the position of a pop-up such as an error message. The component sets the starting position of the pop-up either in absolute coordinates (with the starting position calculated from the top left corner of the screen) or relative either to the active window or to the client space.

In C, this is a deferred component.

Refer to Set_Window_Message for a description of the component that typically uses this component to position the message relative to the window.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.

In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

For 3270 terminals, the absolute and relative positions are identical. State the values for Top_Position and Left_Position in workstation graphic pixel coordinates. The component translates the graphic pixel values into 3270 row-column coordinates based on a display of 24 rows, 18 pixels high, and 80 columns, 8 pixels wide. Row and column coordinates map to pixel coordinates in even multiples of row and column grid sizes. Absolute pixel coordinates of (50, 16), map to row-column coordinates of (3, 2), where 18 divides into 50 three times after rounding up to an integer, and 8 divides evenly into 16 twice. The component positions the upper left of the pop-up message in the third row, second column of the terminal display.

The values the Top_Position and Left_Position fields can vary depending on the value in the Poptype field. If the Poptype field is set to 1 or 3, the value in the Top_Position field can range from +0 to +336 and Left_Position can range from +0 to +640. If the Poptype field is set to 2, the value in the Top_Position field can range from -336 to +336 and Left_Position can range from -640 to +640. However, if the requested location places the pop-up message outside the terminal screen area, an error condition is encountered, and the Return_Code field in the output view is set to zero (failure).

Syntax

Here is the syntax of Set_PopUp_Position component.

Set_PopUp_Position component syntax

Name	`Set_PopUp_Position`
System Identifier	`CPNLPOS`
Input View Name	`Set_PopUp_Position_I`
Input View Field	`Poptype integer(15)` This field is attached to the system set WINDOW_POSITIONS (SPOPPNL), which has the following values: 1 = Absolute coordinates (0, 0 is the top left of the screen) 2 = Relative to the entire window 3 = Relative to the client area (the panel area inside the window)
Input View Field	`Top_Position integer(15)` Top position (positive or negative allowed)
Input View Field	`Left_Position integer(15)` Left position (positive or negative allowed)
Output View Name	`Set_PopUp_Position_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

The values in the Top_Position and the Left_Position fields of the input view represent workstation graphic pixel coordinates. The Top_Position field of the input view sets the position of the top of the pop-up message. The Left_Position field of the input view sets the position of the left side of the pop-up message.

On a workstation, Top_Position and Left_Position can contain any value. The actual position of the window depends on the resolution of the monitor. There is no restriction on the window's position; it can appear outside the visible portion of the window.

Example: Set_Popup_Position

This example repositions the pop-up to the top left corner of the screen.

```
map 1 to POPTYPE of SET_POPUP_POSITION_I
map 0 to TOP_POSITION of SET_POPUP_POSITION_I
map 0 to LEFT_POSITION of SET_POPUP_POSITION_I
use component SET_POPUP_POSITION
```

Set_Push_Color

This section presents the purpose, the syntax, the usage, and how the Set_Push_Color component is supported. See See Examples: Set_Push_Color for an usage example.

Purpose

Use this component to set the color of the background, text, or border of push buttons. The push button text set in the PUSH_TEXT field identifies the target push button.

In C, this is a deferred omponent .

Unlike most other component fields, the PUSH_TEXT field is case sensitive. You must enter text exactly as it appears on the push button. However, do not include any mnemonic indicators (& or ~) or function key names. For example, if the field appears as Help F1, type Help into this field.

⚠ Because of a Windows restriction, use of this component is ignored in applications executed in Microsoft Windows.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients. For the Field_Attr input view field, the reset value 12 is not supported.

In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to See Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Set_Push_Color component.

Set_Push_Color component syntax | |

Name	`Set_Push_Color`
System Identifier	`CPSHCOL`
Input View Name	`Set_Push_Color_I`

Input View Field	`Push_Text character(30)` Push-button text (not system identifier).
Input View Field	`Push_Attr integer(15)` Push-button property to be colored. This field is attached to the system set See WINDOW_OBJECT_ATTRIBUTES (SATTR), which has the following values: • 0 = Background, 1 = Text, 2 = Border • 10 = Resets background to color originally set in Window Painter • 11 = Resets text to color originally set in Window Painter • 12 = Resets border to color originally set in Window Painter The value in the Attr_Color field is ignored when you reset the color.
Input View Field	`Attr_Color integer(15)` Color to make the selected property. This field is attached to the system set See WINDOW_OBJECT_COLORS (SCOLORS), which has the following values: 0 = White, 1 = Yellow, 2 = Blue 3 = Red, 4 = Not supported, 5 = Not supported 6 = Green, 7 = Magenta, 8 = Black 9 = Aqua
Output View Name	`Set_Push_Color_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set See RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Examples: Set_Push_Color

This example specifies the push button named *Continue* as the target of the color change.

```
map 'Continue' to PUSH_TEXT of SET_PUSH_COLOR_I
use component SET_PUSH_COLOR
```

This example maps the value TEXT in the system set WINDOW_OBJECT_ATTRIBUTES to the input view of the component, specifying the text of the *Continue* push button as the target of the color change.

```
map TEXT in WINDOW_OBJECT_ATTRIBUTES to PUSH_ATTR of SET_PUSH_COLOR_I
use component SET_PUSH_COLOR
```

This example maps the value BLUE in the system set WINDOW_OBJECT_COLORS to the input view of the component, specifying that the text of the *Continue* push button to be displayed in blue.

```
map BLUE in WINDOW_OBJECT_COLORS to ATTR_COLOR of SET_PUSH_COLOR_I
use component SET_PUSH_COLOR
```

Set_Push_Mode

This section presents the purpose, the syntax, the usage, and how the Set_Push_Mode component is supported. See Examples: Set_Push_Mode for an usage example.

Purpose

Use this component to change the mode of a push button. The specified push button can be made either visible and enabled, visible and disabled, or invisible. The output Return_Code is set to 1 for success and 0 for failure. The push button text set in the Push_Text field identifies the target push button.

In C, this is a *deferred component* .

⚠ Unlike most other component fields, the Push_Text field is case sensitive. You must enter text exactly as it appears on the push button. However, do not include any mnemonic indicators (& or ~) or function key names. For example, if the field appears as Help F1, type Help into this field.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe.
Refer to 3270 Converse Supported Interface Components for more information.

As push buttons are implemented on the 3270 terminal device as function keys, the message INVALID FUNCTION REQUEST is displayed when the 3270 end user selects a disabled push button.

Syntax

Here is the syntax of Set_Push_Mode component.

Set_Push_Mode component syntax

Name	`Set_Push_Mode`
System Identifier	`CPSHMOD`
Input View Name	`Set_Push_Mode_I`
Input View Field	{{Push_Text character(30)} Push button text (not system identifier).
Input View Field	`Push_Mode integer(15)` Push button mode. This field is attached to the system set WINDOW_OBJECT_MODES (SMODE), which has the following values: 0 = Invisible, disabled 1 = Visible, disabled 2 = Visible, enabled 3 = For mainframe: Invisible, enabled (3270 Converse only) 3 = For Java and C: Visible, read-only
Output View Name	`Set_Push_Mode_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Examples: Set_Push_Mode

This example specifies the push button named **Continue** as the target of the mode change.

```
map 'Continue' to PUSH_TEXT of SET_PUSH_MODE_I
use component SET_PUSH_MODE
```

This example maps the value INVISIBLE in the system set WINDOW_OBJECT_MODES to the input view of the component, specifying the Continue push button to be made invisible.

```
map INVISIBLE in WINDOW_OBJECT_MODES to PUSH_MODE of SET_PUSH_MODE_I
use component SET_PUSH_MODE
```

Set_Virtual_Listbox_Size

This section presents the purpose, the syntax, the usage, and how the Set_Virtual_Listbox_Size component is supported. See Example: Set_Virtual_Listbox_Size for an usage example.

Purpose

Use this component only with a multicolumn list box to define a virtual list box to implement smooth scrolling. To implement smooth scrolling for a single-column list box, use a multicolumn list box with a single, non-editable column.

The SET_VIRTUAL_LISTBOX_SIZE component is intended to be called repeatedly when scrolling through a single set of data. Do not reset anything about the multicolumn list box state at the time of the component call.

In C, this is a deferred component .

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

The 3270 Converse implementation of this system component requires the system to allocate internal storage when any action is performed on a field within the virtual list box. To minimize the amount of storage allocated, this system component fails when the application attempts to create a virtual list box larger than the initial size. Therefore, your ability to increase the virtual list box size may be limited in 3270 Converse.

Syntax

Here is the syntax of Set_Virtual_Listbox_Size component.

Set_Virtual_Listbox_Size component syntax

Name	`Set_Virtual_Listbox_Size`
System Identifier	`CLSTSIZ`
Input View Name	`Set_Virtual_Listbox_Size_I`
Input View Field	`View_Long_Name character(30)` Specifies the view that defines the virtual list box

Input View Field	`Number_Of_Records integer(15)` Defines the size of the virtual list box
Input View Field	`Elevator_Position integer(15)` The virtual number of the first list box view occurrence
Output View Name	`Set_Virtual_Listbox_Size_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Virtual Listbox Size

There is an important distinction between the size of the list box, the list box view, and the virtual list box. The physical list box contains the instances of the occurring field presented to the end user. The list box view data structure contains the instances of the occurring field that can be presented to the end user without returning an OUT OF RANGE condition to the invoking rule. The virtual list box construct contains the instances of the occurring field that are virtually available to the end user.

For example, the physical list box can be defined to contain 10 occurrences, the list box view data structure to contain 100 occurrences, and the virtual list box to contain 1,000 occurrences. The virtual list box usually defines the number of field occurrences that can be retrieved from the database. If no virtual list box is defined, the component assumes that the list box view data structure defines the bounds of the list box.

Virtual Elevator Position

In addition to setting the size of the virtual list box, the this component also sets the virtual elevator position. This virtual elevator position refers to the virtual occurrence number of the data that reside in the first occurrence of the list box view data structure. For example, the virtual elevator position is set to 350 if the first occurrence of the list box view data structure contains the data that represent the 350th occurrence of the virtual list box. You must reset the virtual elevator position each time the data contained in the list box view data structure changes. To set the virtual elevator position, populate the Elevator_Position parameter of the Set_Virtual_Listbox_Size_I input view with the correct value.

After you define a virtual list box with this component, all field occurrences input to or output from list box converse components are in virtual occurrence values.

Example: Set_Virtual_Listbox_Size

This example sets a virtual listbox size.

```
map 'NC_TABLE_SL' to VIEW_LONG_NAME of SET_VIRTUAL_LISTBOX_SIZE_I
map ELEVATOR_POSITION of NC_TABLECOMPONENT_WD of NC_TABLECOMPONENT_W
    to ELEVATOR_POSITION of SET_VIRTUAL_LISTBOX_SIZE_I
map NUMBER_OF_RECORDS of NC_TABLECOMPONENT_WD of NC_TABLECOMPONENT_W
    to NUMBER_OF_RECORDS of SET_VIRTUAL_LISTBOX_SIZE_I
use component SET_VIRTUAL_LISTBOX_SIZE
```

Set_Window_Message

This section presents the purpose, the syntax, the usage, and how the Set_Window_Message component is supported. See Examples: Set_Window_Message for an usage example.

Purpose

Use this component to display an error message in a window (similar to Set_Field_Message). However, this component does not highlight a specific field that is in error. When the window is invoked, the error message is displayed in a window. This window is modeless and remains visible until the user selects the *OK* push button. A window can display only one message at a time.

In C, this is a deferred component.

Refer to Set_PopUp_Position for the component that you can use to position the message.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients. In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information. Also, the message window is modal in Java, so the user must explicitly close this window to go back to the calling window.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for information on other components.

For 3270 Converse applications executing on the mainframe, the message specified by the parameters set in the input view is edited with the text arguments and displayed on line 22 of the 3270 device, or if it is longer than 79 characters, in a scrollable pop-up message displayed at the bottom of the screen, expanding upward as more lines are needed.

Any message text that cannot be displayed is replaced by periods on the display. This component can manage up to 32K bytes of message text data for all the messages of a set. Because it displays messages longer than 79 characters in a bordered pop-up message, only 72 bytes of message text data per line can be displayed on the 3270 terminal device. Be careful when creating message text, because the insertion of arguments can cause the line to be truncated. The 3270 Converse version of the component does not use OK_BUTTON_YN.

Syntax

Here is the syntax of Set_Window_Message component.

Set_Window_Message component syntax

Name	`Set_Window_Message`
System Identifier	`CPNLMSG`
Input View Name	`Set_Window_Message_I`
Input View Field	`Window_Long_Name character(30)` This field is required to specify the window where the message is displayed.
Input View Field	`Text_Arg1 character(18)` First text input in error message.
Input View Field	`Text_Arg2 character(18)` Second text input in error message.
Input View Field	`Text_Arg3 character(18)` Third text input in error message.
Input View Field	`Message_Set_Name character(8)` The implementation name of the reference file without the REF extension. This field specifies the implementation name of the set that contains the error message.
Input View Field	`Text_Code integer(15)` A number indicating which message is to be used from the error set. This field specifies the error message within the set to be displayed.
Input View Field	`OK_Button_YN integer(15)` This field is attached to the set OK_YES_NO (SOKYNO). You do not need to supply any value in this field when you use this component.
Output View Name	`Set_Window_Message_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Usage

You can construct up to three customized inputs by giving values to Text_Arg1, Text_Arg2, and Text_Arg3. These strings are then inserted into the generic error message selected from the error message set. The percent character (%) serves as a place holder in the messages for the Text_Arg inputs. Thus %1 is the place holder for Text_Arg1, %2 for Text_Arg2, and %3 for Text_Arg3. The at sign character (@) is also recognized as a place holder when converting older applications, but do not use it when designing new applications. For example, the message

```
%1, please try another automobile type. The one you chose does not exist.
```

can be customized by mapping a field containing the user's name into Text_Arg1. Thus, the user's name appears in the "%1" position.

Examples: Set_Window_Message

This example specifies PASSENGER_INQUIRY as the window where the message is to be displayed.

```
map 'PASSENGER_INQUIRY' to WINDOW_LONG_NAME of SET_WINDOW_MESSAGE_I
use component SET_WINDOW_MESSAGE
```

This example specifies the set with the implementation name ERRMSGS as the set from which to retrieve the messages to be displayed.

```
map 'ERRMSGS' to MESSAGE_SET_NAME of SET_WINDOW_MESSAGE_I
use component SET_WINDOW_MESSAGE
```

This example specifies the message corresponding to the value NOT_ON_FILE in the set ERROR_MESSAGES is to be displayed. Note that this must be the set whose implementation name was mapped to MESSAGE_SET_NAME above. The value mapped is the symbol on the Set *contains* Value relationship, which you can view in the project hierarchy.

```
caseof WINDOW_RETCODE of CUSTOMER_INQUIRY
   case 'SEARCH'
      use rule RETRIEVE_CUSTOMER_RECORD
      if RETURN_CODE <> SUCCESS in RETURN_CODES
      map NOT_ON_FILE in ERROR_MESSAGES to TEXT_CODE of SET_WINDOW_MESSAGE_I
use component SET_WINDOW_MESSAGE
```

This example specifies the value in the field CUSTOMER_LAST_NAME is to be displayed along with the error message.

```
map CUSTOMER_LAST_NAME of CUSTOMER to TEXT_ARG1 of SET_WINDOW_MESSAGE_I
use component SET_WINDOW_MESSAGE
```

Set_Window_Position

This section presents the purpose, the syntax, the usage, and how the Set_Window_Position component is supported. See Example: Set_Window_Position for an usage example.

Purpose

Use this component to set the position of a non-nested window. The screen coordinates 0,0 indicate the upper left corner of the screen. Each unit is one pixel.

In C, this is a deferred component.

On a workstation, Top_Position and Left_Position can contain any value. The actual position of the window depends on the resolution of the monitor. There is no restriction on the window's position; it can appear outside the visible portion of the window.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Set_Window_Position component.

Set_Window_Position component syntax

Name	`Set_Window_Position`
System Identifier	`CBCKPOS`
Input View Name	`Set_Window_Position_I`
Input View Field	`Top_Position integer(15)` Top coordinate for window (positive and negative allowed)
Input View Field	`Left_Position integer(15)` Left coordinate for window (positive and negative allowed)
Output View Name	`Set_Window_Position_O`
Output View Field	{{Return_Code integer(15) }} This field is attached to the system set RETURN_CODES (SRETURN),which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Set_Window_Position

This example sets the upper left corner of the next window conversed to be 10 pixels down from the top of the screen, and 10 pixels to the right of the left side of the screen.

```
map 10 to TOP_POSITION of SET_WINDOW_POSITION_I
map 10 to LEFT_POSITION of SET_WINDOW_POSITION_I
use component SET_WINDOW_POSITION
```

Set_Window_Timeout

This section presents the purpose, the syntax, the usage, and how the Set_Window_Timeout component is supported. See Example: Set_Window_Timeout for an usage example.

Purpose

Use this component to return control to a rule if a window it has conversed does not receive user input within a specified time period.

If there is no activity on the window for the duration specified in the Number_Of_Seconds field, a TIMEOUT return code is sent to the rule. Number_Of_Seconds can be a maximum of 65,525 seconds (approximately 18.2 hours). Any value greater than this or less than 0 causes the component request to fail. A Number_Of_Seconds value of 0 clears any previous settings for a TIMEOUT.

In C, this is a deferred component.

⚠️ If a foreground panel is visible and receives a TIMEOUT code, control is passed to the nested rule. When you return to the rule that called the nested rule, WINDOW_RETCODE of the pre-defined system view HPS_EVENT_VIEW is not set automatically. You must set the output view of the nested rule and test it in the calling rule.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Set_Window_Timeout component.

Set_Window_Timeout component syntax

Name	`Set_Window_TimeOut`
System Identifier	`CPNLTME`
Input View Name	`Set_Window_TimeOut_I`
Input View Field	`Window_Long_Name character(30)`
Input View Field	`Number_Of_Seconds integer(31)` Number of seconds after which window is to time out
Output View Name	`Set_Window_TimeOut_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Set_Window_Timeout

In this example, if CUSTOMER_DETAILS is the next window conversed, and if no activity occurs within 5 seconds, control returns to the calling rule along with the HPS_WIN_TIMEOUT event. (If CUSTOMER_DETAILS is not the next window to be conversed, the component has no effect.)

```
map 5 to NUMBER_OF_SECONDS of SET_WINDOW_TIMEOUT_I
map 'Customer Details' to WINDOW_LONG_NAME of SET_WINDOW_TIMEOUT_I
use component SET_WINDOW_TIMEOUT
```

Set_Window_Title

This section presents the purpose, the syntax, the usage, and how the Set_Window_Title component is supported. See Example: Set_Window_Title for an usage example.

Purpose

Use this component to set the title (that is, the caption) of a window. With this component, you can create a panel title different than the panel name.

In C, this is a deferred component .

⚠ If you leave the Title_Length field blank, or if you enter a 0, the component displays the full 50 characters of the Caption_Text field.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.

In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

⚠ You must have a value in the window title (caption) for this component to work in a thin (HTML) client.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Set_Window_Title component.

Set_Window_Title component syntax

Name	`Set_Window_Title`
System Identifier	`CSETTIT`
Input View Name	`Set_Window_Title_I`
Input View Field	`Caption_Text character(50)` Text of the title or caption for the window.
Input View Field	`Title_Length integer(15)` Length of the title or caption for the window.
Output View Name	`Set_Window_Title_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Set_Window_Title

This example changes the title of the next window conversed to Customer Details.

```
map 'Customer Details' to CAPTION_TEXT of SET_WINDOW_TITLE_I
map 0 to TITLE_LENGTH of SET_WINDOW_TITLE_I
use component SET_WINDOW_TITLE
```

Set_Window_Title_Ex

This section presents the purpose, the syntax, the usage, and how the Set_Window_Title_Ex component is supported. See Example: Set_Window_Title_Ex for an usage example.

Purpose

Use this component to set a very long title (that is, the caption) of a window. With this component, you can create a panel title different than the panel name. This component is an extension to Set_Window_Title component.

In C, this is a deferred component .

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.
In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

> ⚠ You must have a value in the window title (caption) for this component to work in a thin (HTML) client.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Set_Window_Title_Ex component.

Set_Window_Title_Ex component syntax

Name	`Set_Window_Title_Ex`
System Identifier	`CSETTIT`
Input View Name	`Set_Window_Title_Ex_I`
Input View Field	`Caption character(1000)` Caption or title for window
Input View Field	`Caption_Length integer(15)` Length of caption or title for window. If you leave the Caption_Length field blank, or if you enter a 0, the component displays the full 50 characters of the Caption field.
Output View Name	`Set_Window_Title_Ex_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Set_Window_Title_Ex This example changes the title of the next window conversed to Customer Details.

```
map 'Customer Details' to CAPTION of SET_WINDOW_TITLE_EX_I
map 0 to CAPTION_LENGTH of SET_WINDOW_TITLE_EX_I
use component SET_WINDOW_TITLE_EX
```

Show_Help_Topic

This section presents the purpose, the syntax, the usage, and how the Show_Help_Topic component is supported. See Example: Show_Help_Topic for an usage example.

Purpose

Use this component with Set_Help_File_Name to create a help facility for your application. This component displays the help window for the key word specified and is functional only when the help file name associated with the application has been specified with the Set_Help_File_Name component.

In C, this is a deferred component.

> ⚠ You can use this component only with the native GUI help file that is already linked with the application.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients.
In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Show_Help_Topic component.

Show_Help_Topic component syntax

Name	`Show_Help_Topic`
System Identifier	`CSHOHLP`
Input View Name	`Show_Help_Topic_I`
Input View Field	`Help_Keyword character(50)` Keyword of topic to be displayed
Output View Name	`Show_Help_Topic_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Show_Help_Topic

This example maps BUTTON0 as the keyword for the help facility. BUTTON0 is also the system identifier (HPSID) of the push button.

```
converse window AETUTOR_PUSHBUTTON_OO
caseof WINDOW_RETCODE of AETUTOR_PUSHBUTTON_OO
   case 'BUTTON0'
      map 'BUTTON0' to HELP_KEYWORD of SHOW_HELP_TOPIC_I
use component SHOW_HELP_TOPIC
```

Show_Window_Message

This section presents the purpose, the syntax, the usage, and how the Show_Window_Message component is supported. See Example: Show_Window_Message for an usage example.

Purpose

Use this component to display a particular message when a window is conversed. This component displays a pop-up message when it may be impossible to use a rule to converse the window. For example, you may want to display the prompt "Wait a moment please" while fetching data from the mainframe. The message disappears the next time the window is conversed. Also see Set_Window_Message.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Show_Window_Message component.

Show_Window_Message component syntax

Name	`Show_Window_Message`
System Identifier	`CIMMERR`
Input View Name	`Show_Window_Message_I`
Input View Field	`Window_Long_Name character(30)` This field is required to specify the window where the message is displayed.
Input View Field	`Text_Arg1 character(18)` First text input in error message.
Input View Field	`Text_Arg2 character(18)` Second text input in error message.
Input View Field	`Text_Arg3 character(18)` Third text input in error message.
Input View Field	`Message_Set_Name character(8)` The implementation name of the reference file without the REF extension. This field specifies the implementation name of the set that contains the error message.
Input View Field	`Text_Code integer(15)` A number indicating which message is to be used from the error set. This field specifies the error message within the set to be displayed.
Input View Field	`OK_Button_YN integer(15)` This field is attached to the system set OK_YES_NO (SOKYNO). You do not need to supply any value in this field when you use this component.
Output View Name	`Show_Window_Message_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Usage

You use this component the same as Set_Window_Message, except, of course, for the name of the component.

You can construct up to three customized inputs by giving values to Text_Arg1, Text_Arg2, and Text_Arg3. These strings are then inserted into the generic error message selected from the error message set. The percent character (%) serves as a place holder in the messages for the Text_Arg inputs. Thus %1 is the place holder for Text_Arg1, %2 for Text_Arg2, and %3 for Text_Arg3. The at sign character (@) is also recognized as a place holder when converting older applications, but do not use it when designing new applications. For example, the message

```
%1, please try another automobile type. The one you chose does not exist.
```

can be customized by mapping a field containing the user's name into Text_Arg1. Thus, the user's name appears in the "%1" position.

Example: Show_Window_Message

This example shows a window message.

```
map WARNING_ERROR_MSG in CUSTOMER_ERROR_WD to TEXT_CODE of SHOW_WINDOW_MESSAGE_I
map 'ZADZQYK' to MESSAGE_SET_NAME of SHOW_WINDOW_MESSAGE_I
map 'Invalid ' to TEXT_ARG1 of SHOW_WINDOW_MESSAGE_I
map 'desc.txt' to TEXT_ARG2 of SHOW_WINDOW_MESSAGE_I
map 'DISPLAY_ERROR_MSG' to WINDOW_LONG_NAME of SHOW_WINDOW_MESSAGE_I
map YES in OK_YES_NO to OK_BUTTON_YN of SHOW_WINDOW_MESSAGE_I
use component SHOW_WINDOW_MESSAGE
```

Sound_3270_Alarm

This section presents the purpose, the syntax, the usage, and how the Sound_3270_Alarm component is supported. See Example: Sound_3270_Alarm for an usage example.

Purpose

Use this component to cause the audible alarm on the 3270 device to beep when the next window is conversed. The default is to not sound the alarm unless there is an error or warning that has been detected by the 3270 Converse runtime code.

⚠ This component is only valid in the 3270 Converse environment and functions for all 3270 devices that support the alarm feature.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Sound_3270_Alarm component.

Sound_3270_Alarm component syntax

Name	`Sound_3270_Alarm`
System Identifier	`CSNDALM`
Input View Name	This component does not have an input view.
Output View Name	This component does not have an output view.

See the examples under Set_Window_Message for more details.

Example: Sound_3270_Alarm

This example sounds the alarm.

```
use component Sound_3270_Alarm
```

Cache_Object

This section presents the purpose, the syntax, the usage, and how the Cache_Object component is supported. See Example: Cache_Object for an usage example.

Purpose

Use this component to preload rules or user components (or both) into the cache at execution time. The size of the cache is controlled by the RULE_BUFFERS setting in the AE Runtime section of the system initialization file (hps.ini). The setting defaults to 40 on all platforms.

Java Support

This component is not supported for either thick (Java) or thin (HTML) clients.

C# Support

This component is not supported for C# client.

Syntax

Here is the syntax of Cache_Object component.

Cache_Object component syntax

Name	`Cache_Object`
System Identifier	`CCACHE`
Input View Name	`Cache_Object_I`
Input View Field	`Object_Name character(100)` The implementation name of the rule or user component, which will normally default to its system id.
Input View Field	`Object_Type integer(15)` The type of object being cached. • 0 = AppBuilder rule DLL (dynamic link library) • 1 = AppBuilder user component DLL
Input View Field	`Permanent integer(15)` Determines if the object is placed in the cache permanently or temporarily. • 0 = Temporary • 1 = Permanent
Output View Name	`Cache_Object_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Usage

Rules and user components can be added to the cache permanently or temporarily. If you try to load more objects into the cache than it can hold, the component dynamically increases the size of the cache. The cache does not dynamically grow for the rule DLLs (dynamic link libraries) that are loaded using the USE RULE statements. Rules loaded this way are considered to be temporary members of the cache. If a temporary object is the least-used object in the cache, it is removed when the cache is full. Temporary objects are also removed from the cache if room is needed for a permanent object. Objects that have been put into the cache permanently remain there until the application is closed.
Although the code for caching a component can be put into any rule, it is probably best to put it at the beginning of the root Windows workstation rule, so that all of the loading is done when the application is started.
Remember these restrictions when caching objects:

- If you increase the size of the cache and increase the value of RULE_BUFFERS, your application uses more memory as more and more objects are stored in the cache.
- If you are using a user component that allocates global memory, this component must be permanently cached.

Example: Cache_Object

In this example, the rule DLL is permanently cached.

```
map 'ZAYYES' to Object_Name of Cache_Object_I
map 0 to Object_Type of Cache_Object_I
map 1 to Permanent of Cache_Object_I
use component Cache_Object
```

Clear_Field_Messages

This section presents the purpose, the syntax, the usage, and how the Clear_Field_Messages component is supported. See Example: Clear_Field_Messages for an usage example.

Purpose

Use this component to reset the fields in a given panel to a non-error condition. Specifically, this component removes the highlighting from any fields in error and clears any error messages that the *Set_Field_Message* component has displayed.
In C, this is a deferred component.

Java Support

This component is supported for thick (Java) clients but not thin (HTML) clients and supports only edit fields.
In the Java client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Syntax

Here is the syntax of Clear_Field_Message component.

Clear_Field_Message component syntax

Name	`Clear_Field_Messages`
System Identifier	`CCLRFLD`
Input View Name	`Clear_Field_Messages_I`
Input View Field	`Window_Long_Name character(30)`
Output View Name	`Clear_Field_Messages_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred components, so Return_Code is set to 1 unless a system-level error occurs. When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.
Return	`V_CLEAR_FIELD_MESSAGES`

Related Components

Use this component with Set_Field_Message .

Example: Clear_Field_Messages

This example specifies GET_CUSTOMER_ID_FROM_NAME as the window on which to reset the fields.

```
map 'GET_CUSTOMER_ID_FROM_NAME' to
Window_Long_Name of Clear_Field_Messages_I
use component Clear_Field_Messages
```

Clear_Selected_Fields

This section presents the purpose, the syntax, the usage, and how the Clear_Field_Fields component is supported. See Example: Clear_Selected_Fields for an usage example.

Purpose

Use this component to deselect the fields in the list boxes and tables of a given window. If the user has selected any fields from the panel, after using this component and conversing the window, none of the fields in the window remain selected.
In C, this is a deferred component.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients.
In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

Syntax

Here is the syntax of Clear_Selected_Fiekds component.

Clear_Selected_Fiekds component syntax

Name	`Clear_Selected_Fields`
System Identifier	`CCLRSLT`
Input View Name	`Clear_Selected_Fields_I`
Input View Field	`Window_Long_Name character(30)`
Output View Name	`Clear_Selected_Fields_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs.When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

⚠ You can deselect fields only in list boxes with this component.

Example: Clear_Selected_Fields

This example specifies SELECT_CUSTOMER as the window on which to clear selected fields.

```
map 'SELECT_CUSTOMER' to WINDOW_LONG_NAME of CLEAR_SELECTED_FIELDS_I
use component CLEAR_SELECTED_FIELDS
```

Clear_Window_Changes

This section presents the purpose, the syntax, the usage, and how the Clear_Window_Changes component is supported. See Example: Clear_Window_Changes for an usage example.

Purpose

Use this component to reset the status of window objects to the status they had when the window was originally created. For example, if you use *Set_Field_Color* to change the text color of a field to catch a user's attention, this component returns the field to its previous color and resets the window to its original position.
In C, this is a deferred component.

Java Support

This component is supported for both thick (Java) clients and thin (HTML) clients. This component must be enabled in the appbuilder.ini file for it to work in a Java environment. Refer to the *Configuring Communications Guide* for instructions on using the Management Console to edit an INI file. Refer to Environment Details for other specific differences in the Java environment.
In the Java client and thin client, this component acts on the current window and not on the next conversed window. This is because there is no deferred component functionality in Java. Refer to Java and C# Support for Deferred Components for more information.

C# Support

This component is supported for C# client.

3270 Converse Support

This component is supported in a CICS or IMS online application (3270 Converse) on the mainframe. Refer to 3270 Converse Supported Interface Components for more information.

Environment Details

Depending on which environment (C, C#, or Java), different sets of properties may be reset. The properties of objects in a window (typically defined using Window Painter and modifiable in the properties panel) are summarized in Object properties reset by Clear_Window_Changes. In Java, Position and Size are only reset for the window object itself and not for the objects in that window.

Object properties reset by Clear_Window_Changes

Property of Object	C (Windows) Client	Java Client andHTML Thin Client
Background Color	Yes	Yes
Border Color	Yes	No
Checked (menu items)	Yes	Yes
Close Text (Window)	Yes	No
Editable	Yes	Yes
Enable/Disable	Yes	Yes
Field Altered Status	Yes	Yes
Field Error Status	Yes	Yes
Field Picture	Yes	No
Font	No	No
Foreground Color	Yes	Yes
Link	Yes	No
MCLB Scroll Position	Yes	Yes
Position	Yes	No, except for window object

Selection (ComboBox, ListBox and Table(MCLB))	Yes	Yes
ShortHelp	No	No
Size	Yes	No, except for window object
Text	Yes	No
Visible	Yes	Yes
Virtual MCLB Size	Yes	Yes

If you use other components that affect the properties of Window objects at execution time (for example, HPS_Limit_Scroll_Size) and then run the Clear_Window_Changes component, Clear_Window_Changes resets the Window object properties.

Syntax

Here is the syntax of Clear_Window_Changes component.

Clear_Window_Changes component syntax

Name	`Clear_Window_Changes`
System Identifier	`CCLRPNL`
Input View Name	`Clear_Window_Changes_I`
Input View Field	`Window_Long_Name character(30)`
Output View Name	`Clear_Window_Changes_O`
Output View Field	`Return_Code integer(15)` This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure. In C, this is a deferred component, so Return_Code is set to 1 unless a system-level error occurs.When the CONVERSE WINDOW statement executes, the contents of the input view are checked. If an error is detected at this time, a message box is displayed that indicates the error.

Example: Clear_Window_Changes

This example specifies SELECT_CUSTOMER as the window on which to clear changes.

```
map 'SELECT_CUSTOMER' to WINDOW_LONG_NAME of CLEAR_WINDOW_CHANGES_I
use component CLEAR_WINDOW_CHANGES
```

Smooth Scrolling with Components

Using the elevator and scrolling arrows in list boxes should be transparent to end users. Several of the interface system components are used together to provide this smooth scrolling. Use the smooth scrolling components only with multicolumn list boxes (MCLBs). To implement smooth scrolling for single-column list box, use a MCLB object and make it a single, non-editable column. The following are two examples:

- Smooth Scrolling: Converse Rules Example
- Smooth Scrolling: Event-Driven Rules Example

For more information on smooth scrolling, refer to the DataRequired event and the Table object in the *ObjectSpeak Reference Guide*.
The examples in this topic demonstrate how to use these system components for smooth scrolling:

- Get_Elevator_Position
- Get_Listbox_Window_Sizes
- Set_First_Visible_Occurrence
- Set_Virtual_Listbox_Size

Smooth Scrolling: Converse Rules Example

This example shows an MCLB smooth scrolling for converse rules (C-RULES). This example has a List Window Display Rule (AB_SMOOTH_SCROLL_DIS) and a Fetch Rule (AB_MERULE_SQL_FET). Refer to Fetch Rule Used in Examples.

```
*>-------- Smooth Scrolling Example for Converse Rules --------<*
*                                                               *
*  This example shows how to smooth scroll the MCLB (or table)  *
*  for CONVERSE RULES(C-RULES).                                 *
*                                                               *
*  Rule: AB_SMOOTH_SCROLL_DIS - List window display rule        *
*  Child rule :AB_MERULE_SQL_FET - Fetch rule                   *
*                                                               *
*  This displays the rule information from the                  *
*  personal repository database table (MERULE).                 *
*--------------------------------------------------------------<*

dcl
    L_BACK_BUFFER_AMT smallint;
    L_FIRST_VISIBLE_OCC smallint;
    L_FIRST_SCROLL_OCC smallint;
enddcl

*>initialize the variables<*
perform InitializeProc

*>Get the first block of data and display<*
perform GetNextBlock

*> Converse loop <*
do  while EVENT_SOURCE of HPS_EVENT_VIEW <> 'EXIT'

    *> Converse the window <*
    converse window AB_SMOOTH_SCROLL

    caseof EVENT_NAME of HPS_EVENT_VIEW
        case 'HPS_LB_OUTRANGE'

            use component GET_ELEVATOR_POSITION

            map ELEVATOR_POSITION of GET_ELEVATOR_POSITION_O
            to L_FIRST_VISIBLE_OCC
            map (L_FIRST_VISIBLE_OCC - L_BACK_BUFFER_AMT)
            to L_FIRST_SCROLL_OCC

            *> Calculate starting point for fetch making sure it
            doesn't begin with an invalid record number<*

            *> Lower boundary condition <*
            if L_FIRST_SCROLL_OCC < 1
                map 1 to L_FIRST_SCROLL_OCC
            endif

            *> Upper boundary condition <*
            if L_FIRST_SCROLL_OCC > (AB_NUM_OF_RECS of
                AB_MERULE_SQL_FET_O - SCROLLABLE_OCCURS of
                GET_LISTBOX_WINDOW_SIZES_O + 1)

                map (AB_NUM_OF_RECS of AB_MERULE_SQL_FET_O -
                    SCROLLABLE_OCCURS of GET_LISTBOX_WINDOW_SIZES_O +1)
                to L_FIRST_SCROLL_OCC
            endif

            *>Get Next Block of data and display<*
            perform GetNextBlock

        case 'HPS_LB_BOTTOM'
```

```
            *> Reset current elevator position <*
            map (AB_NUM_OF_RECS of AB_MERULE_SQL_FET_O -
                L_FIRST_VISIBLE_OCC + 1)
            to L_FIRST_SCROLL_OCC

    endcase

enddo

*>-----------------------------------------------------------*
* 1. Fetch Next block of data from the database and map to
* MCLB occurring view
* 2. Sets the new virtual mclb sizes
* 3. Sets the first visible row to be shown
*-----------------------------------------------------------<*
proc GetNextBlock

    *> The First scrollable occurrence is passed to the fetch rule as
        starting point for retrieving data. <*
    map L_FIRST_SCROLL_OCC
    to AB_ELEV_POS of AB_MERULE_SQL_FET_I

    use rule AB_MERULE_SQL_FET

    map AB_MERULE_D of AB_MERULE_SQL_FET_O
    to AB_SMOOTH_SCROLL_OCC of AB_SMOOTH_SCROLL_W

    *> Set Virtual List Box Size <*
    map AB_NUM_OF_RECS of AB_MERULE_SQL_FET_O
    to NUMBER_OF_RECORDS of SET_VIRTUAL_LISTBOX_SIZE_I
    map L_FIRST_SCROLL_OCC to ELEVATOR_POSITION of
    SET_VIRTUAL_LISTBOX_SIZE_I

    use component SET_VIRTUAL_LISTBOX_SIZE

    *> Set First Visible Occurrence <*
    map L_FIRST_VISIBLE_OCC
    to FIELD_OCCUR of SET_FIRST_VISIBLE_OCCURRENCE_I

    use component SET_FIRST_VISIBLE_OCCURRENCE

endproc

*>------------------------------------------------*
* This proc is called first to initialize the
* component views and local variables
*------------------------------------------------<*
proc InitializeProc

    *>--- initialize listbox component input views --<*
    map 'AB_SMOOTH_SCROLL_OCC'
    to VIEW_LONG_NAME of SET_VIRTUAL_LISTBOX_SIZE_I
    map 'AB_SMOOTH_SCROLL_OCC'
    to VIEW_LONG_NAME of SET_FIRST_VISIBLE_OCCURRENCE_I
    map 'AB_SMOOTH_SCROLL_OCC'
    to VIEW_LONG_NAME of GET_LISTBOX_WINDOW_SIZES_I
    map 'AB_SMOOTH_SCROLL_OCC'
    to VIEW_LONG_NAME of GET_ELEVATOR_POSITION_I

    *>----- initialize elevator position ---------<*
    map 1 to L_FIRST_VISIBLE_OCC
    map 1 to L_FIRST_SCROLL_OCC

    converse window AB_SMOOTH_SCROLL nowait

    *> After first converse find out number of list lines and compute
    back buffer -- the number of record back from first visible record
    to include in the view. This is so if the user goes back a little
    and does not have to fetch data outside of his view. <*
    use component GET_LISTBOX_WINDOW_SIZES
    *> The following algorithm can calculate the back buffer
    amount. Note: the amount of back buffer may not exceed the
    occurrences
    <*
    map (SCROLLABLE_OCCURS of GET_LISTBOX_WINDOW_SIZES_O
    - VISIBLE_OCCURS of GET_LISTBOX_WINDOW_SIZES_O) / 2
    to L_BACK_BUFFER_AMT

endproc
```

```
*>---------- End Smooth Scrolling for Converse Rules --------<*
```

Smooth Scrolling: Event-Driven Rules Example

This example shows an MCLB smooth scrolling for event-driven rules (Java client and thin HTML client). This example has a list window display rule (AB_SMOOTH_SCROLL_ED_DIS) and uses the same fetch rule (AB_MERULE_SQL_FET) as the previous example. Refer to Fetch Rule Used in Examples.

```
*>------ Smooth Scrolling Example for Event-Driven Rules ------<*
*                                                              *
*  This example shows how to smooth scroll the MCLB (or table)  *
*  for event driven rules (JAVA/HTML-RULES).                    *
*                                                              *
*  Rule: AB_SMOOTH_SCROLL_ED_DIS - List window display rule     *
*                                                              *
*  This displays the rule information from the                 *
*  personal repository database table (MERULE).                *
*--------------------------------------------------------------<*
*
* The following entries in appbuilder.ini are required:
* [DB]
* DB_ACCESS=LOCAL
* DBMS_PREFIX=jdbc:db2:
* JDBC_DRIVER=COM.ibm.db2.jdbc.app.DB2Driver
* DBNAME=SDCFS_LR
* USERID=HPSFWY
* PASSWORD=<password>
*
*------------------------------------------------------------------<*

dcl
    BackBufferAmt smallint;
    FirstVisibleOcc smallint;
    FirstScrollOcc smallint;
enddcl

//----------------------------------------------------------
// Fetch the database for next block of data and
// map it to the occurring view of MCLB.
//----------------------------------------------------------
proc GetNextBlock(fromIndex smallint)
    //fetch next block from the index from index
    map fromIndex
    to AB_ELEV_POS of AB_MERULE_SQL_FET_I
    use rule AB_MERULE_SQL_FET

    //map to occuring view of mclb
    map AB_MERULE_D of AB_MERULE_SQL_FET_O
    to AB_SMOOTH_SCROLL_ED_OCC of AB_SMOOTH_SCROLL_ED_W
endproc

//----------------------------------------------------------
// Update the MCLB Virtual Sizes
// and First visible row.
//----------------------------------------------------------
proc UpdateMCLBDisplay
    //set virtual listbox size
    MERULE_MCLB.setVirtualListBoxSize(FirstScrollOcc,
                                      AB_NUM_OF_RECS of
                                      AB_MERULE_SQL_FET_O)
    //Sets First Visible Row
    MERULE_MCLB.setFirstVisibleRow(FirstVisibleOcc)
endproc

//----------------------------------------------------------
```

```
// This Procedure gets called when a window is initialized
// the first time before it is shown.
// Here:
// 1. Initialize the local variables
// 2. Fetch and map first block of data
// 3. Update the MCLB size and display
//-----------------------------------------------------------
proc InitEventProc for initialize object AB_SMOOTH_SCROLL_ED
(e object pointer to InitializeEvent)

    //initialize elevator positions
    map 1 to FirstVisibleOcc
    map 1 to FirstScrollOcc

    //Compute back buffer -- the number of record back from first
    // visible record to include in the view. This is so if the
    // user goes back a little he doesn't have to fetch data
    //outside of his view.
    map (MERULE_MCLB.ScrollableOccurs - MERULE_MCLB.VisibleOccurs) / 2
    to BackBufferAmt

    //Fetch next block of data and map
    GetNextBlock(FirstScrollOcc)

    //update the MCLB Display
    UpdateMCLBDisplay
endproc

//-----------------------------------------------------------
// This procedure is called every time user scrolls pass
// the current virtual limit.
// Here:
// 1. Calculate first scrollable occurrence
// 2. Calculate first visible occurrence
// 3. Fetch and map the next block of data
// 4. Update the MCLB size and display
//-----------------------------------------------------------
proc DataEventProc for DataRequired object MERULE_MCLB
(e object pointer to DataRequiredEvent)
    if e.TypeString = 'OutOfRange'
        //get the current elevator positin
        map e.TopVirtualRow to FirstVisibleOcc
        map (FirstVisibleOcc - BackBufferAmt) to FirstScrollOcc

        //Calculate starting point for fetch making sure it
        //doesn't begin with an invalid record number
        //Lower boundary condition
        if FirstScrollOcc < 1
            map 1 to FirstScrollOcc
        endif

        // Upper boundary condition
        if FirstScrollOcc > (AB_NUM_OF_RECS of
            AB_MERULE_SQL_FET_O - MERULE_MCLB.ScrollableOccurs + 1)

            map (AB_NUM_OF_RECS of AB_MERULE_SQL_FET_O -
                 MERULE_MCLB.ScrollableOccurs + 1)
            to FirstScrollOcc
        endif

        //fetch next block and map to mclb occ view
        GetNextBlock(FirstScrollOcc)

        //update virtual listbox sizes
        UpdateMCLBDisplay
    endif
endproc

//-----------------------------------------------------------
// Click Event Procedure for the push button EXIT
// Terminate the window here
//---------------------------------------------------------
proc ExitProc for Click object EXIT
(e object pointer to ClickEvent)
    AB_SMOOTH_SCROLL_ED.terminate
endproc
```

```
*>------- End Smooth Scrolling for Event-Driven Rules -----<*
```

Fetch Rule Used in Examples

```
*>----------------------------------------------------------*
* Rule : AB_MERULE_SQL_FET - fetch rule                     *
* This rule uses Personal Repository table MERULE.          *
* Create a user named HPSFWY and give full permission to    *
* your Personal Repository database.                        *
*---------------------------------------------------------<*

dcl
    L_COUNT smallint;
    L_ROWCOUNT smallint;
    L_OCCURSIZE smallint;
    L_SEARCH_FIELD like AB_SEARCH_NAME;
enddcl

*> map input data to local variables <*
map AB_SEARCH_NAME of AB_MERULE_SEARCH_KEY of AB_MERULE_SQL_FET_I
to L_SEARCH_FIELD
map occurs(AB_MERULE_D)
to L_OCCURSIZE

*>---------------------------------------------------------
* Calculate the number of records meeting search criteria.
* This necessary to set virtual listbox size
*---------------------------------------------------------<*
sql asis
    select count(*)
    into :L_ROWCOUNT
    from HPSFWY.MERULE A
    where A.NAME > :L_SEARCH_FIELD
    and A.LATEST = 'X'
    and A.DELETION = ' '
endsql

*> map number of records to fetch out put view <*
map L_ROWCOUNT
to AB_NUM_OF_RECS of AB_MERULE_SQL_FET_O

*> Selects all records meeting search criteria into cursor <*
sql asis
    declare MERULE1 cursor for
    select A.SHORTNAME,
           A.NAME,
           A.REMOTEMAINTENANCED,
           A.REMOTEMAINTAINEDBY,
           A.PROJECT
    from HPSFWY.MERULE A
    where A.NAME > :L_SEARCH_FIELD
        and A.LATEST = 'X'
        and A.DELETION = ' '
    order by A.NAME
endsql

sql asis
    open MERULE1
endsql

*> if cursor fails, return failure<*
if SQLCODE of SQLCA <> 0
    map FAILURE in RETURN_CODES
    to AB_RET_CODE of AB_MERULE_SQL_FET_O
    return
```

```
endif

*> The window display rule passed the elevator position to the fetch
rule. All record positions selected before the elevator position
should be discarded. <*
do from 1 to (AB_ELEV_POS of AB_MERULE_SQL_FET_I - 1)
    sql asis
        fetch MERULE1
        into :AB_MERULE_DATA.AB_MERULE_SHORTNAME,
             :AB_MERULE_DATA.AB_MERULE_NAME,
             :AB_MERULE_DATA.AB_MERULE_REM_MAINT_DT,
             :AB_MERULE_DATA.AB_MERULE_REM_MAINT_BY,
             :AB_MERULE_DATA.AB_MERULE_PROJECT
    endsql
enddo

*> Fill the occuring view <*
do from 1 to L_OCCURSIZE index L_COUNT
    sql asis
        fetch MERULE1
        into :AB_MERULE_DATA.AB_MERULE_SHORTNAME,
             :AB_MERULE_DATA.AB_MERULE_NAME,
             :AB_MERULE_DATA.AB_MERULE_REM_MAINT_DT,
             :AB_MERULE_DATA.AB_MERULE_REM_MAINT_BY,
             :AB_MERULE_DATA.AB_MERULE_PROJECT
    endsql

    while SQLCODE of SQLCA = 0
        map AB_MERULE_DATA
        to AB_MERULE_D of AB_MERULE_SQL_FET_O(L_COUNT)
enddo

map SQLCODE of SQLCA
to AB_RET_CODE of AB_MERULE_SQL_FET_O

sql asis
    close MERULE1
endsql
```

```
*>------------ End of Fetch Rule for Examples -----------<*
```

DDE System Components

Dynamic Data Exchange (DDE) components are for C Language applications only. DDE is a message protocol for data exchange between two processes. DDE client components (DDECs) hide the complexities of the DDE protocol from the developer. The system supports DDECs for DOS/Windows but not for AIX or mainframe-based applications.
DDE client components summarizes the supported DDE client components.

DDE client components

Component	DDE message	Description
DDE_ADVISE	WM_DDE_ADVISE	Create a permanent channel
DDE_EXECUTE	WM_DDE_EXECUTE	Request server to execute a command
DDE_INITIATE	WM_DDE_INITIATE	Initialize a session
DDE_POKE	WM_DDE_POKE	Send data to an item
DDE_REQUEST	WM_DDE_REQUEST	Request data for an item
DDE_TERMINATE	WM_DDE_TERMINATE	Terminate a session
DDE_UNADVISE	WM_DDE_UNADVISE	Break channel created by DDE_ADVISE

DDE_ADVISE

This section presents the purpose, the syntax, and an usage example for the DDE_ADVISE component. See Example: DDE_ADVISE.

Purpose

This component creates a "hot link" for an item. It requests that the server/topic sends updates on a given item whenever data changes. The "hot link" remains active until the server connection is terminated or until a DDE_UNADVISE is issued on the item. Updates to the item are placed in the view pointed to by the VIEW_LONG_NAME and VIEW_QUALIFIER.

Syntax

Here is the syntax of DDE_ADVISE component.

DDE_ADVISE component syntax

Name	DDE_ADVISE
System Identifier	CDDEADV
Type	IMMEDIATE
Input View Name	DDE_ADVISE_I
Input View Field	SERVER_HANDLE integer(31) Server/topic handle of DDE link
Input View Field	VIEW_LONG_NAME character(30) Root view to be updated
Input View Field	VIEW_QUALIFIER character(256) Path for the root view as shown in its data link
	SERVER_ITEM character(50) The server to which data is sent
Output View Name	DDE_ADVISE_O

Output View Field	RETURN_CODE integer(15) This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: DDE_ADVISE

AE_DDE_HANDLE is a user-defined field containing the handle to a server/topic saved from the use of DDE_INITIATE. AE_DDE_DEMO is the name of the root view and AE_DDE_DEMO_DATA[0] the name of the subview with its first occurrence. The data link these two views specify points to the location in the application hierarchy that accepts the data. The string "R1C1:R5C2" represents the Excel items to which the "hot link" is established. Any changes to those items are reflected automatically in the AE_DDE_DEMO_DATA view.

```
map 'AE_DDE_DEMO' to VIEW_LONG_NAME of DDE_ADVISE_I
map 'AE_DDE_DEMO_DATA[0]' to VIEW_QUALIFIER of DDE_ADVISE_I
map 'R1C1:R5C2' to SERVER_ITEM of DDE_ADVISE_I
map AE_DDE_HANDLE to SERVER_HANDLE of DDE_ADVISE_I
use component DDE_ADVISE
```

It makes more sense to use DDE_ADVISE over DDE_REQUEST when there is a need for "real-time" data. For instance, assume you want to extract stock price data from another application. If you use DDE_ADVISE, any changes in the other application are reflected automatically in your application.

DDE_EXECUTE

This section presents the purpose, the syntax, and an usage example of the DDE_EXECUTE component. See Example: DDE_EXECUTE.

Purpose

This component requests the server to execute a command string. The command is a text string in a format the server defines.

Syntax

Here is the syntax of DDE_EXECUTE component.

DDE_EXECUTE component syntax

Name	**DDE_EXECUTE**
System Identifier	CDDEXEC
Type	IMMEDIATE
Input View Name	DDE_EXECUTE_I
Input View Field	SERVER_HANDLE integer(31) Server/topic handle of the DDE link
Input View Field	COMMAND_FIELD character(512) String for the server to execute
Output View Name	DDE_EXECUTE_O
Output View Field	RETURN_CODE integer(15) This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: DDE_EXECUTE

AE_DDE_HANDLE is a user-defined field that contains the handle to server/topic saved from the DDE_INITIATE. The command string tells Excel to select a range of cells on the current spreadsheet, graph them, and maximize the graph.

```
map AE_DDE_HANDLE to SERVER_HANDLE of DDE_EXECUTE_I
map '[SELECT("R1C1:R5C2,R1C1")][NEW(2)][FULL]' to COMMAND_FIELD of DDE_EXECUTE_I
use component DDE_EXECUTE
```

DDE_INITIATE

This section presents the purpose, the syntax, and an usage example of the DDE_INITIATE component. See Example: DDE_INITIATE.

Purpose

This component initializes a DDE link for a given topic and server. If the DDE server can support the requested topic, it returns a handle to the server/topic, SERVER_HANDLE. You must save this handle, which all the other DDE components use to communicate with the server on that topic. The DDE server application defines the topic and server names.

Syntax

Here is the syntax of DDE_INITIATE component.

DDE_INITIATE component syntax

Name	**DDE_INITIATE**
System Identifier	CDDEINI
Type	IMMEDIATE
Input View Name	DDE_INITIATE_I
Input View Field	SERVER_NAME character(50) Name of server
Input View Field	SERVER_TOPIC character(50) Name of topic
Output View Name	DDE_INITIATE_O
Output View Field	RETURN_CODE integer(15) This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.
Output View Field	SERVER_HANDLE integer(31) Server/topic handle of DDE link

Example: DDE_INITIATE

In this example, a connection is made to a Microsoft Excel spreadsheet called "Sheet1". If the connection is successful, a server/topic handle is sent back. The user saves the returned SERVER_HANDLE, which is used in all subsequent communications with Excel about the spreadsheet. The AE_DDE_HANDLE is a user-defined field for storing the handle.

```
map 'EXCEL' to SERVER_NAME
map 'SHEET1' to SERVER_TOPIC
use component DDE_INITIATE
map SERVER_HANDLE of DDE_INITIATE_O to AE_DDE_HANDLE
```

DDE_POKE

This section presents the purpose, the syntax, and an usage example of the DDE_POKE component. See Example: DDE_POKE.

Purpose

This component sends data to the server/topic for a specified item. This is a one-time request; if the data changes, you must use this component again to send the changed data. The data is sent from the view pointed to by the VIEW_LONG_NAME and VIEW_QUALIFIER.

Syntax

Here is the syntax of DDE_POKE component.

DDE_POKE component syntax

Name	**DDE_POKE**

System Identifier	CDDESND
Type	IMMEDIATE
Input View Name	DDE_POKE_I
Input View Field	SERVER_HANDLE integer(31) Server/topic handle of the DDE link
Input View Field	VIEW_LONG_NAME character(30) View to be updated
Input View Field	VIEW_QUALIFIER character(256) Path for the view as shown in its data link
Input View Field	SERVER_ITEM character(50) Destination of the data
Output View Name	DDE_POKE_O
Output View Field	RETURN_CODE integer(15) This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: DDE_POKE

AE_DDE_HANDLE is a user-defined field containing the handle to a server/topic saved from the use of DDE_INITIATE. AE_DDE_DEMO is the name of the root view and AE_DDE_DEMO_DATA[0] the name of the subview with its first occurrence. The data link these two views specify points to the data being sent. The string "R1C1:R5C2" represents the Excel items to which the data is sent.

```
map 'AE_DDE_DEMO' to VIEW_LONG_NAME of DDE_POKE_I
map 'AE_DDE_DEMO_DATA[0]' to VIEW_QUALIFIER of DDE_POKE_I
map 'R1C1:R5C5' to SERVER_ITEM of DDE_POKE_I
map AE_DDE_HANDLE to SERVER_HANDLE of DDE_POKE_I
use component DDE_POKE
```

DDE_REQUEST

This section presents the purpose, the syntax, and an usage example of the DDE_REQUEST component. See Example: DDE_REQUEST.

Purpose

This component requests data from the server/topic for an item. This is a one-time request; if the data changes, you must use this component again to retrieve the changed data. The data updates the view pointed to by the VIEW_LONG_NAME and VIEW_QUALIFIER.

Syntax

Here is the syntax of DDE_REQUEST component.

DDE_REQUEST component syntax

Name	**DDE_REQUEST**
System Identifier	CDDEREQ
Type	IMMEDIATE
Input View Name	DDE_REQUEST_I
Input View Field	SERVER_HANDLE integer(31) Server/topic handle of DDE link
Input View Field	VIEW_LONG_NAME character(30) Root view to be updated
Input View Field	VIEW_QUALIFIER character(256) Path for the root view as shown in its data link

Input View Field	SERVER_ITEM character(50) The server to which data is sent
Output View Name	DDE_REQUEST_O
Output View Field	RETURN_CODE integer(15) This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: DDE_REQUEST

AE_DDE_HANDLE is a user-defined field containing the handle to a server/topic saved from the use of DDE_INITIATE. AE_DDE_DEMO is the name of the root view and AE_DDE_DEMO_DATA[0] the name of the subview with its first occurrence. The data link these two views specify points to the location in the application hierarchy that accepts the data. The string "R1C1:R5C2" represents the Excel items from which data is retrieved.

```
map 'AE_DDE_DEMO' to VIEW_LONG_NAME of DDE_REQUEST_I
map 'AE_DDE_DEMO_DATA[0]' to VIEW_QUALIFIER of DDE_REQUEST_I
map 'R1C1:R5C2' to SERVER_ITEM of DDE_REQUEST_I
map AE_DDE_HANDLE to SERVER_HANDLE of DDE_REQUEST_I
use component DDE_REQUEST
```

DDE_TERMINATE

This section presents the purpose, the syntax, and an usage example of the DDE_TERMINATE component. See Example: DDE_TERMINATE.

Purpose

This component ends a connection with a server/topic. Once the DDE connection is terminated, the SERVER_HANDLE handle is invalid.

Syntax

Here is the syntax of DDE_TERMINATE component.

DDE_TERMINATE component syntax

Name	**DDE_TERMINATE**
System Identifier	CDDETRM
Type	IMMEDIATE
Input View Name	DDE_TERMINATE_I
Input View Field	SERVER_HANDLE integer(31) Server/topic handle of DDE link
Output View Name	DDE_TERMINATE_O_
Output View Field	*RETURN_CODE integer(15)* This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: DDE_TERMINATE

The AE_DDE_HANDLE is a user-defined field that contains the handle to access the server/topic saved from the DDE_INITIATE.

```
if ( AE_DDE_HANDLE <> 0 )
map AE_DDE_HANDLE to SERVER_HANDLE of DDE_TERMINATE_I
use component DDE_TERMINATE
map 0 to AE_DDE_HANDLE
endif
```

DDE_UNADVISE

This section presents the purpose, the syntax, and an usage example of the DDE_UNADVISE component. See Example: DDE_UNADVISE.

Purpose

This component terminates a "hot link" previously created with a call to DDE_ADVISE. This component tells the server not to send any more updates on this item.

Syntax

Here is the syntax of DDE_UNADVISE component.

DDE_UNADVISE component syntax

Name	**DDE_UNADVISE**
System Identifier	CDDEUDV
Type	IMMEDIATE
Input View Name	DDE_UNADVISE_I
Input View Field	SERVER_HANDLE integer(31) Server/topic handle of the DDE link
Input View Field	SERVER_ITEM character(50) Name of data to unadvise
Output View Name	DDE_UNADVISE_O
Output View Field	RETURN_CODE integer(15) This field is attached to the system set RETURN_CODES (SRETURN), which has the following values: 1 for Success, 0 for Failure.

Example: DDE_UNADVISE

AE_DDE_HANDLE is a user-defined field containing the handle to a server/topic saved from the use of DDE_INITIATE. The string "R1C1:R5C2" represents the Excel items to which the "hot link" is established.

```
map 'R1C1:R5C2' to SERVER_ITEM of DDE_UNADVISE_I
map AE_DDE_HANDLE to SERVER_HANDLE of DDE_UNADVISE_I
use component DDE_UNADVISE
```

System Components for Mainframe, Unix and Java Batch

There are system components that work only on mainframe (host) systems only for online MVS - CICS and IMS. These include:

- Mainframe Batch System Components
- OpenCOBOL for Unix & Java Batch System Components
- IMS Components
- BLOB System Components

Some batch system components and interface system components are supported for UNIX. Refer to OpenCOBOL for Unix & Java Batch System Components for a list of supported system components.

Mainframe Batch System Components

A number of components are available in the repository for execution in the (MVS) mainframe batch environment. You can use them for the following reasons:

- Storage Management Components - for managing storage
- Input and Output Components - for handling QSAM input and output of fixed and variable length records
- Report Writer Components - for generating information for reports
- System Information Components - for retrieving system information

Two batch components, DB2 and CGENV, appear in the enterprise repository but are reserved for internal use. Three components (CDYNPIC, HPDTTM, and HPSETVP) are discussed in the MVS Report Writer. You can use the MVS LOCATE I/O components on either side of the host or the workstation.

The storage management components retain the values of static variables that, because of the dynamic nature of the linkage interface, are cleared each time an entity is used. These components allocate, locate, and free blocks of memory of various sizes from the operating system pools.

The input and output components deal with the related issue of maintaining positioning within sequential files. The functions they perform include opening, reading, writing, and closing files. You may use these components on mainframe.

The system information components retrieve variables from the environment and do not process any input data.

You communicate with these components using standard input and output view passing. By convention, a component's input view is the name of the component with the letter "I" appended. Similarly, the name of its output view is the component's name with the letter "O" appended.

Batch components are written in assembler and are reentrant, so you can realize significant performance improvements by installing them in the link pack area. They have been link-edited using an addressing mode of 31 and a residency mode of 24. I/O components must reside below the 16 MB line as they switch into addressing mode 24 when calling mainframe routines.

Storage Management Components

Storage management is handled by the following components:

- HPGET (as in GETMAIN) and HPGETL
- HPFIND and HPFINDL
- HPFREE (as in FREEMAIN) and HPFREEL

These components allocate, locate, and free blocks of memory of various sizes from the operating system pools. The following sample rule, HPFSR01, illustrates their actions.

This rule first uses HPGET to allocate an area for HPFSC01's static variables. It then uses HPFIND and HPFSC01 repeatedly until HPFSC01 returns a value of 10 in its output view. HPFSC01 establishes addressability to its static variables, increments COUNTER, and returns its value in its output view. Finally, HPFSR01 uses HPFREE to deallocate the block of storage.

HPGET

This section presents the purpose, the syntax, and the usage of the HPGET component.

Purpose

This component allocates blocks of storage of any desired length.

Syntax

Here is the syntax of HPGET component.

HPGET component syntax

Name	HPGET
System Identifier	`HPGET`
Input View Name	`HPGETI`
Input View Field	`KEY character(8)` System identifier of a static variable view
Input View Field	`SIZE integer(15)` Length to be allocated
Output View Name	`HPGETO`
Output View Field	`COMPLETE integer(31)` A return code indicating the status of the request • 0 = Normal • 4 = Duplicate address of storage

Output View Field	`STORAGE integer(31)` Address of storage

Usage

The first field of this component's input view is KEY. This field should be an arbitrary eight-character string that becomes the name used to refer to the block in subsequent calls to the location and deallocation components such as HPFIND and HPFREE. The second field of its input view is SIZE, a small integer field that contains the number of bytes of memory required to satisfy the request. The operating system rounds this number up to the next multiple of eight, if necessary.

HPGET executes a GETMAIN to allocate the required storage plus a fixed-length header that is not addressable by the calling rule. The block is then inserted into an ordered linked list whose address the MAINAREA control block maintains. Then call HPFIND or HPFREE to search this list. The storage resides below the 16-megabyte line and is initialized to binary zeroes.

The first field of the output view is the long integer COMPLETE. A completion code of 0 in this field indicates success and a code of 4 indicates a block with the same key is already on the list. The second field is the long integer STORAGE. This field normally contains the address of the block allocated. In the case of a duplication, it contains the address of the existing block of the same name.

HPGETL

This section presents the purpose, the syntax, and the usage of the HPGETL component.

Purpose

Similar to HPGET, this component allocates blocks of storage of any desired length. However, while the field KEY stores the system name of the static variable view in the input view of HPGET, the field KEY_LONG_NM stores the view's name in the input view of HPGETL.

Syntax

Here is the syntax of HPGETL component.

HPGETL component syntax

Name	**HPGETL**
System Identifier	`HPGETL`
Input View Name	`HPGETLI`
Input View Field	`KEY_LONG_NM character(30)` View name of a static variable view
Input View Field	`SIZE integer(15)` Length to be allocated
Output View Name	`HPGETO` (note this is not `HPGETLO`)
Output View Field	`COMPLETE integer(31)` • 0 = Normal • 4 = Duplicate address of storage
Output View Field	`STORAGE integer(31)` Address of storage

Usage

The first field of the input view of this component is KEY_LONG_NM. This field specifies the name of the static variable view referred to in subsequent calls to the location and deallocation components such as HPFIND and HPFREE. The second field of its input view is SIZE, a small integer field that contains the number of bytes of memory required to satisfy the request. The mainframe operating system rounds this number up to the next multiple of eight, if necessary.

This component executes a GETMAIN to allocate the required storage plus a fixed-length header that the calling rule cannot address. The block is then inserted into an ordered linked list whose address the MAINAREA control block maintains. Then call HPFIND or HPFREE to search this list to locate or deallocate a block. The storage resides below the 16-megabyte line and is initialized to binary zeroes.

The first field of the output view of this component is the long integer COMPLETE. A completion code of 0 in this field indicates success and a code of 4 indicates that a block with the same key is already on the list. The second field is the long integer STORAGE. This field normally contains the address of the block allocated. In the case of a duplication, it contains the address of the existing block of the same name.

HPFIND

This section presents the purpose, the syntax, and the usage of the HPFIND component.

Purpose

This component locates a block of storage already acquired through a use of HPGET.

Syntax

Here is the syntax of HPFIND component.
HPFIND component syntax

Name	**HPFIND**
System Identifier	`HPFIND`
Input View Name	`HPFINDI`
Input View Field	`KEY character(8)` System identifier of a static variable view
Output View Name	`HPFINDO`
Output View Field	`COMPLETE integer(31)` • 0 = Normal • 4 = Not found
Output View Field	`STORAGE integer(31)` Address of storage

Usage

The character field KEY is the only field in the input view of the component. KEY should be the same eight-character string specified in a previous allocation request using the HPGET component.

HPFINDL

This section presents the purpose, the syntax, and the usage of the HPFINDL component.

Purpose

This component has the same functionality as HPFIND and locates a block of storage already acquired with HPGET. The only difference is in the input view: KEY_LONG_NM can be up to 30 characters long.

Syntax

Here is the syntax of HPFINDL component.

HPFINDL component syntax

Name	**HPFINDL**
System Identifier	`HPFINDL`
Input View Name	`HPFINDLI`
Input View Field	`KEY_LONG_NM character(30)` View name of a static variable view
Output View Name	`HPFINDO` (note this is not `HPFINDLO`)
Output View Field	`COMPLETE integer(31)` • 0 = Normal • 4 = Not found

Output View Field	`STORAGE integer(31)` Address of storage

Usage

This component is functionally equivalent to the HPFIND component. The only difference is in the input views' field. For the HPFIND component, the KEY field uses the system name of the static variable view that provides input to the component. This component uses the name, stored in the field KEY_LONG_NM, of the static variable view.

HPFREE

This section presents the purpose, the syntax, and the usage of the HPFREE component.

Purpose

This component searches the linked list of blocks for one with a matching key. When the desired block is found, it is removed from the list and a FREEMAIN is executed to return it to the free storage pool. If the desired block is not found, no action is taken.

Syntax

Here is the syntax of HPFREE component.

HPFREE component syntax

Name	**HPFREE**
System Identifier	`HPFREE`
Input View Name	`HPFREEI`
Input View Field	`KEY character(8)` System identifier of a static variable view
Output View Name	`HPFREEO`
Output View Field	`COMPLETE integer(31)` • 0 = Normal • 4 = Not found

Usage

The only field of the output view of this component is the long integer COMPLETE. A completion code of (0) in this field indicates success; a code of (4) indicates that the block with the specified key did not exist on the list.

HPFREEL

This section presents the purpose, the syntax, and the usage of the HPFREEL component. See Example - Use of Storage Management Components for an usage example.

Purpose

Similar to the HPFREE component, the HPFREEL component searches the linked list of blocks for one with a matching key. When the desired block is found, it is removed from the list and a FREEMAIN is executed to return it to the free storage pool. If the desired block is not found, no action is taken. However, the HPFREE component stores the system name of the static variable view in the field KEY of the input view, but HPFREEL stores the name of the static variable view in the KEY_LONG_NM field.

Syntax

Here is the syntax of HPFREEL component.

HPFREEL component syntax

Name	**HPFREEL**
System Identifier	`HPFREEL`

Input View Name	HPFREELI
Input View Field	KEY_LONG_NM character(30) View name of a static variable view
Output View Name	HPFREEO (note this is not HPFREELO)
Output View Field	COMPLETE integer(31) • 0 = Normal • 4 = Not found

Example - Use of Storage Management Components

This example shows the use of storage management components.

```
*>*****************************************************\**<*
*> RULE HPFSR01 <*
*>*\*****************************************************\**<*
map 'STATIC' TO KEY OF HPGETI
map 8 TO SIZE OF HPGETI
use component HPGET
do while COUNTER OF HPFSC010 <> 10
    map 'STATIC' TO KEY OF HPFINDI
    use component HPFIND
    map STORAGE OF HPFINDO TO STORAGE OF HPFSC01I
    use component HPFSC01
enddo
map 'STATIC' TO KEY OF HPFREEI
use component HPFREE
```

Only subcomponents can use storage that these components obtain. The following uses the previously defined rule.

```
*******************************************************/
/\* COMPONENT HPFSCO1 \*/
*//*\***************************************************/
HPFSC01: PROCEDURE(COMMADDR) OPTIONS(MAIN)
    DCL COMMADDR PTR;
    DCL 1 COMMAREA BASED (COMMADDR) UNALIGNED,
    %INCLUDE HPSCOMM
    DCL 1 INPUT_VIEW        BASED(IV_PTR),
        3 STORAGE           PTR;
    DCL 1 OUTPUT_VIEW       BASED(OV_PTR),
        3 COUNTER FIXED     BIN(15);
    DCL 1 STATIC_VARIABLES  BASED(STORAGE),
        3 STATIC_COUNTER    FIXED BIN(15);
    STATIC_COUNTER = STATIC_COUNTER \+1;
    COUNTER = STATIC_COUNTER;
    END HPFSC01;
```

Input and Output Components

There are two sets of four components each that handle QSAM input and output of fixed and variable length records. These components maintain sequential files and positioning within them. The managed files can be fixed length or variable length and blocked or unblocked, but you must define them before referencing them. The functions they perform include opening, reading, writing, and closing sequential files. In addition, the SET_HPSBATCH_RETURN_CODE component allows you to set the value of the return code that the HPSBATCH program returns.

The first set of components uses the MOVE I/O mode. Use these components only on the mainframe:

- HPOPEN (opens a sequential file)
- HPREAD (reads from a sequential file)
- HPWRITE (writes to a sequential file)
- HPCLOSE (closes a sequential file)

The second set of components performs the same functionality as the first set, but uses the LOCATE I/O mode. This mode passes pointers to data, instead of the data themselves, and hence is more efficient:

- HPS_OPEN_FILE_LOCATE_MODE (opens a sequential file)
- HPS_READ_FILE_LOCATE_MODE (reads from a sequential file)
- HPS_WRITE_FILE_LOCATE_MODE (writes to a sequential file)
- HPS_CLOSE_FILE_LOCATE_MODE (closes a sequential file)

There is also a component for flushing data records stored in I/O buffers:

- HPS_TRUNC_FILE_LOCATE_MODE

There is also a component that sets the value of the return code that the HPSBATCH program returns:

- SET_HPSBATCH_RETURN_CODE

JCL Considerations

To minimize coding requirements for rules development and provide maximum flexibility at runtime, the input and output components make no assumptions about the files being processed, other than that the records are either fixed or variable length. The records can be blocked or unblocked but must be defined. This means that the file's characteristics are taken from an existing data set's labels or from the JCL if a data set is created. Remember, you must explicitly override unacceptable defaults on DD cards or TSO ALLOCATE commands.

HPOPEN

This section presents the purpose, the syntax, and the usage of the HPOPEN component.

Purpose

This component initiates processing a sequential mainframe data set through the standard QSAM interface. Storage for a mainframe data set control block (DCB) is allocated by using a suballocation of dynamic storage that was acquired by the batch driver at startup. The DCB is then initialized and opened for either input or output depending on the value of the MODE field.

Syntax

Here is the syntax of HPOPEN component.

HPOPEN component syntax

Name	**HPOPEN**
System Identifier	`HPOPEN`
Input View Name	`HPOPENI`
Input View Field	`DDNAME character(8)` Identifies the JCL file in a corresponding TSO ALLOCATE command or DD card
Input View Field	`MODE character(8)` Can be one of two values: • `INPUT` indicates data is taken from the data set (default) • `OUTPUT` indicates data is passed to the data set
Output View Name	`HPOPENO`
Output View Field	`COMPLETE integer(31)` • 0 = Successful • 4 = Could not be opened
Output View Field	`DCB integer(31)` Address of the open data set control block (DCB)

Usage

Missing DD cards for the files concerned are the most common cause of open failures. Do not try to process a file that does not open successfully. In the case of an open failure, the DCB contents are meaningless.

HPREAD

This section presents the purpose, the syntax, and the usage of the HPREAD component.

Purpose

This component reads records from a sequential mainframe data set through the standard QSAM interface. It executes a GET to read a single record from the input file. The record data is then copied from an internal buffer below the 16-megabyte line to the component's output view.

Syntax

Here is the syntax of HPREAD component.

HPREAD component syntax

Name	**HPREAD**
System Identifier	`HPREAD`
Input View Name	`HPREADI`
Input View Field	`DCB integer(31)` Address of the data set control block (DCB) obtained through use of the HPOPEN component
Output View Name	`HPREADO`
Output View Field	`COMPLETE integer(31)` • 0 = Successful • 4 = End of file
Output View Field	`SIZE integer(15)` Record length
Output View Field	`RECORD integer(1920)` Data read from the data set

Usage

Do not try to read records from a file when the end of the file has been reached. The contents of the RECORD field are meaningless at end of file.

The SIZE field contains the length of the record read when processing variable length records. It does not include the length of the record descriptor word. The contents of the SIZE field are the logical record length of the input data set when processing fixed-length records.

The length of the RECORD field in the repository imposes an upper bound on the logical record length of the files this component can read. If this limit is insufficient, you can use the field RECORDL, which allows for a larger character field. However, you must then prepare the HPREADO output view and the HPREAD component.

HPWRITE

This section presents the purpose, the syntax, and the usage of the HPWRITE component.

Purpose

This component writes records to a sequential mainframe data set through the standard QSAM interface. It copies the record data from the input view to an internal buffer below the 16-megabyte line. A PUT is then executed to write the single record to the output file.

Syntax

Here is the syntax of HPWRITE component.

HPWRITE component

Name	**HPWRITE**
System Identifier	`HPWRITE`
Input View Name	`HPWRITEI`

Input View Field	`DCB integer(31)` Address of the data set control block (DCB) obtained through use of the HPOPEN component.
Input View Field	`SIZE integer(15)` Variable record length.
Input View Field	`RECORD character(1920)` Data to be written to the set.
Output View Name	`HPWRITEO`
Output View Field	`COMPLETE integer(31)` • 0 = Success • 4 = Could not be written

Usage

The SIZE field must contain the length of the record to be written when processing variable-length records. It should not include the length of a record descriptor word. The contents of the SIZE field are ignored when processing fixed-length records. (The DCB contains the necessary record length information.)

The length of the RECORD field in the repository imposes an upper bound on the logical record length of the files that this component can write. If this limit is insufficient, increase the size of the field and prepare the HPWRITEI view again. You must make a corresponding change to the RECORDL equate in the HPWRITE component and then prepare the component again.

HPCLOSE

This section presents the purpose, the syntax, and the usage of the HPCLOSE component.

Purpose

This component concludes processing a sequential mainframe data set through the standard QSAM interface. It closes the file, causing the mainframe operating system to take the appropriate actions for the data set, as specified in the JCL. Storage for the data set control block (DCB) is then deallocated by using a suballocation of dynamic storage that was acquired by the batch driver at startup.

Syntax

Here is the syntax of HPCLOSE component.

HPCLOSE component syntax

Name	**HPCLOSE**
System Identifier	`HPCLOSE`
Input View Name	`HPCLOSEI`
Input View Field	`DCB integer(31)` Address of the DCB obtained through the use of the HPOPEN component.
Output View Name	`HPCLOSEO`
Output View Field	`COMPLETE integer(31)` • 0 = Success This is the only value returned. Any condition causing a close to fail usually causes the batch job to abend (for example, a missing DD statement).

Usage

Do not try to process a closed file without first reopening it.

HPS_OPEN_FILE_LOCATE_MODE

This section presents the purpose, the syntax, and the usage of the HPS_OPEN_FILE_LOCATE_MODE component.

Purpose

This component initiates processing a sequential mainframe data set through the standard QSAM interface. Storage for a mainframe data set control block (DCB) is allocated by this component. The DCB is then initialized and opened for either input or output depending on the value of the MODE field.

This component is functionally equivalent to HPOPEN, except that this component uses LOCATE I/O mode instead of MOVE I/O mode, and hence is more efficient.

Syntax

Here is the syntax of HPS_OPEN_FILE_LOCATE_MODE component.

HPS_OPEN_FILE_LOCATE_MODE component syntax

Name	**HPS_OPEN_FILE_LOCATE_MODE**
System Identifier	`CHPOPEN`
Input View Name	`HPS_OPEN_FILE_LOCATE_MODE_I`
Input View Field	`DDNAME character(8)` Identifies the JCL file in a corresponding TSO ALLOCATE command or DD card
Input View Field	`MODE character(8)` Can be one of two values: • `INPUT` indicates data is taken from the data set (default) • `OUTPUT` indicates data is passed to the data set See OpenCOBOL for Unix & Java Batch System Components for MODE used for OpenCOBOL for Unix & Java Batch.
Output View Name	`HPS_OPEN_FILE_LOCATE_MODE_O`
Output View Field	`COMPLETE integer(31)` The following codes are returned on the workstation: • 0 = Successful • 4 = File could not be opened or invalid mode parameter The following codes are returned on the mainframe: • 0 = Successful • 4 = File could not be opened
Output View Field	`LOCATE_MODE_DCB character(4)` Address of the open data set control block (DCB).

Usage

Missing DD cards for the files concerned are the most common cause of open failures. Do not try to process a file that does not open successfully. In the case of an open failure, the DCB contents are meaningless.

HPS_READ_FILE_LOCATE_MODE

This section presents the purpose, the syntax, and the usage of the HPS_READ_FILE_LOCATE_MODE component.

Purpose

This component reads records from a sequential data set through the standard QSAM interface. It executes a GET to read a single record from the input file. The record data is then copied from an internal buffer below the 16-megabyte line to view specified by the LOCATE_RECORD field.

This component is functionally equivalent to HPREAD, except that this component uses LOCATE I/O mode instead of MOVE I/O mode, and hence is more efficient.

Syntax

Here is the syntax of HPS_READ_FILE_LOCATE_MODE component.

HPS_READ_FILE_LOCATE_MODE component syntax

Name	**HPS_READ_FILE_LOCATE_MODE**
System Identifier	`CHPREAD`
Input View Name	`HPS_READ_FILE_LOCATE_MODE_I`
Input View Field	`VIEW_LONG_NAME character(30)` The name of the view to be read
Input View Field	`SIZE_LONG integer(31)` Record length in bytes
Input View Field	`LOCATE_MODE_DCB character(4)` Address of the data set control block (DCB) obtained through use of the HPS_OPEN_FILE_LOCATE_MODE component
Input View Field	`LOCATE_RECORD character(8)` Location of the view structure where the component should place the data after it has read it from the record. Use the LOC function to supply the view location. (Refer to the *Rules Language Reference Guide* .)
Output View Name	`HPS_READ_FILE_LOCATE_MODE_O`
Output View Field	`COMPLETE integer(31)` The following codes are returned on the workstation: • 0 = Successful • 4 = End of file reached • 8 = Invalid DCB or the read operation failed The following codes are returned on the mainframe: • 0 = Successful • 4 = End of file reached.
Output View Field	`SIZE_LONG integer(31)` Record length

Usage

It is imperative that the value supplied to the SIZE_LONG field in the component input view not exceed the size of the view into which the record is being read. It is recommended that the function SIZEOF(viewname) be used to determine the value to supply to the component. For example:
`map SIZEOF(viewname) to SIZE_LONG of HPS_READ_FILE_LOCATE_MODE_I`

HPS_WRITE_FILE_LOCATE_MODE

This section presents the purpose, the syntax, and the usage of the HPS_WRITE_FILE_LOCATE_MODE component.

Purpose

This component writes records to a sequential mainframe data set through the standard QSAM interface. It copies the record data specified by the LOCATE_RECORD field of the input view to an internal buffer below the 16-megabyte line. A PUT is then executed to write the single record to the output file.

This component is functionally equivalent to HPWRITE, except that this component uses LOCATE I/O mode instead of MOVE I/O mode, and therefore is more efficient.

Syntax

Here is the syntax of HPS_WRITE_FILE_LOCATE_MODE component.

HPS_WRITE_FILE_LOCATE_MODE component syntax

Name	**HPS_WRITE_FILE_LOCATE_MODE**
System Identifier	`CHPWRIT`

Input View Name	`HPS_WRITE_FILE_LOCATE_MODE_I`
Input View Field	`VIEW_LONG_NAME character(30)` Name of the view to be written
Input View Field	`SIZE_LONG integer(31)` Record length in bytes
Input View Field	`LOCATE_MODE_DCB character(4)` Address of the data set control block (DCB) obtained through use of the HPS_OPEN_FILE_LOCATE_MODE component
Input View Field	*LOCATE_RECORD character(8)* Location of the view structure where the component gets the data for writing to the record. Use the LOC function to supply the location of a view. (Refer to the *Rules Language Reference Guide* .)
Output View Name	`HPS_WRITE_FILE_LOCATE_MODE_O`
Output View Field	COMPLETE `integer` (31) The following codes are returned on the workstation: • 0 = Success • 8 = Invalid DCB or the write operation failed The following codes are returned on the mainframe: • 0 = Success • 4 = Could not be written

Usage

The SIZE_LONG field must contain the length of the record to be written when processing variable-length records. It should not include the length of a record descriptor word. The contents of the SIZE_LONG field are ignored when processing fixed-length records. (The DCB contains the necessary record length information.)

HPS_CLOSE_FILE_LOCATE_MODE

This section presents the purpose, the syntax, and the usage of the HPS_CLOSE_FILE_LOCATE_MODE component.

Purpose

This component concludes processing a sequential mainframe data set through the standard QSAM interface. It closes the file, causing to take the appropriate actions for the data set, as specified in the JCL. Storage for the data set control block (DCB) and the file buffer pools is freed by this component.

This component is functionally equivalent to HPCLOSE, except that this component uses LOCATE I/O mode instead of MOVE I/O mode, and hence is more efficient.

Syntax

Here is the syntax of HPS_CLOSE_FILE_LOCATE_MODE component.

HPS_CLOSE_FILE_LOCATE_MODE component syntax

Name	**HPS_CLOSE_FILE_LOCATE_MODE**
System Identifier	`CHPCLOS`
Input View Name	`HPS_CLOSE_FILE_LOCATE_MODE_I`
Input View Field	`LOCATE_MODE_DCB character(4)` Address of the DCB obtained through the use of the HPS_OPEN_FILE_LOCATE_MODE component
Output View Name	`HPS_CLOSE_FILE_LOCATE_MODE_O`

Output View Field	`COMPLETE integer(31)` The following codes are returned on the workstation: • 0 = Success • 4 = Invalid DCB or the close operation failed The following code is returned on the mainframe: • 0 = Success This is the only value returned. Any condition causing a close to fail usually causes the batch job to abend (for example, a missing DD statement).

Usage

Do not try to process a closed file; first open it.

HPS_TRUNC_FILE_LOCATE_MODE

This section presents the purpose, the syntax, and the usage of the HPS_TRUNC_FILE_LOCATE_MODE component.

Purpose

Batch jobs that use the mainframe checkpoint/restart feature to recover from failure can experience a loss of data from files used by rules. If you lose data in this way, you can use this system component to flush data records stored in the I/O buffers. This component issues the mainframe TRUNC macro command for the specified file, which causes the operating system to regard the current buffer as full, allowing a short block to be written to the file.

Syntax

Here is the syntax of HPS_TRUNC_FILE_LOCATE_MODE component.

HPS_TRUNC_FILE_LOCATE_MODE component syntax

Name	**HPS_TRUNC_FILE_LOCATE_MODE**
System Identifier	`CHPTRUN`
Input View Name	`HPS_TRUNC_FILE_LOCATE_MODE_I`
Input View Field	`LOCATE_MODE_DCB character(4)` Address of the DCB obtained through the use of the HPS_OPEN_FILE_LOCATE_MODE component
Output View Name	`HPS_TRUNC_FILE_LOCATE_MODE_O`
Output View Field	`COMPLETE integer(31)` The following codes are returned on the workstation: • 0 = Success • 4 = Invalid DCB or the close operation failed The following code is returned on the mainframe: • 0 = Success This is the only value returned. Any condition causing a close to fail usually causes the batch job to abend (for example, a missing DD statement).

SET_HPSBATCH_RETURN_CODE

This section presents the purpose, the syntax, and the usage of the SET_HPSBATCH_RETURN_CODE component. See Example - Using MOVE I/O Mode for an usage example.

Purpose

This component sets the value of the return code that the HPSBATCH program returns. Executing this component twice within the same rule displays only the last value assigned.

Syntax

Here is the syntax of SET_HPSBATCH_RETURN_CODE component.

SET_HPSBATCH_RETURN_CODE component syntax

Name	**SET_HPSBATCH_RETURN_CODE**
System Identifier	`HPBRETC`
Input View Name	`SET_HPSBATCH_RETURN_CODE_I`
Input View Field	`RETURN_CODE integer(15)` This can be a number from 0 to 4,095

Example - Using MOVE I/O Mode

The following sample rule, HPIOR01, illustrates the actions of the MOVE I/O mode components. This rule copies a fixed or variable length input file to a fixed or variable length output file.

```
*>****************************************************\**<*
*> RULE HPIOR01 <*
*>*\****************************************************\**<*
map 'FILEIN' TO DDNAME OF HPOPENI
map 'INPUT' TO MODE OF HPOPENI
use component HPOPEN
map DCB OF HPOPENO TO DCB OF HPREADI
map 'FILEOUT' TO DDNAME OF HPOPENI
map 'OUTPUT' TO MODE OF HPOPENI
use component HPOPEN
map DCB OF HPOPENO TO DCB OF HPWRITEI
use component HPREAD
do while COMPLETE OF HPREADO = 0
   map RECORD OF HPREADO TO RECORD OF HPWRITEI
   map SIZE OF HPREADO TO SIZE OF HPWRITEI
   use component HPWRITE
   use component HPREAD
enddo
map DCB OF HPREADI TO DCB OF HPCLOSEI
use component HPCLOSE
map DCB OF HPWRITEI TO DCB OF HPCLOSEI
use component HPCLOSE
```

This example shows JCL suitable for executing the sample rule HPIOR01.

```
 *//* Background JCL DD card for fixed length input *//*
//FILEIN DD DISP=SHR,DSN=NA2XX.TEST.FIXED
//\* Background JCL DD card for fixed length output \*//
//FILEOUT DD SYSOUT=*,DCB=(RECFM=F,BLKSIZE=121)
//\* Background JCL DD card for variable length input \*//
//FILEIN DD DISP=SHR,DSN=NA2XX.TEST.VARIABLE
//\* Background JCL DD card for variable length output \*//
//FILEOUT DD SYSOUT=*,DCB=(RECFM=V,BLKSIZE=121)
```

This example shows a CLIST suitable for executing the sample rule HPIOR01.

```
/* Foreground TSO ALLOCATE for fixed length input */
    ATTRIB FIXED RECFM(F)
    ALLOCATE FI(FILEIN) DA(*y) USING(FIXED)
/\* Foreground TSO ALLOCATE for fixed length output \*/
    ATTRIB FIXED RECFM(F)
    ALLOCATE FI(FILEOUT) DA(*y) USING(FIXED)
/\* Foreground TSO ALLOCATE for variable length input \*/
    ATTRIB VARIABLE RECFM(V) BLKSIZE(84) LRECL(80)
    ALLOCATE FI(FILEIN) DA(*y) USING(VARIABLE)
/\* Foreground TSO ALLOCATE for variable length output \*/
    ATTRIB VARIABLE RECFM(V) BLKSIZE(84) LRECL(80)
    ALLOCATE FI(FILEOUT) DA(*y) USING(VARIABLE)
```

Report Writer Components

The Report Writer uses these components:

- CDYNPIC
- HPDTTM
- HPSETVP

CDYNPIC

This section presents the purpose, the syntax, and the usage of the CDYNPIC component.

Purpose

This system component does the following for dynamic picture editing:

- Validates that PIC_FMT_AMT has only a sign or digits, or both.
- PIC_FMT specifies a valid picture. Anything that is accepted at generation time is also accepted at runtime. Exception: Signs are specified with "+" or "-" and not with "S".
- Transforms PIC_FMT_AMT together with PIC_FMT_SCALE into PIC_FMT_RESULT according to PIC specification in PIC_FMT.
- Validates for run-time overflow (return all "*"s).

Report Writer invokes this component behind the scenes. You do not need to invoke it explicitly in your rules to makes dynamic picture editing work. Refer to HPSETVP.

Syntax

This system component with a mainframe batch execution environment accepts as input:

```
01 VRWFORMI
    03 PIC_FMT_AMT PIC 'S9(15)'
    03 PIC_FMT_SCALE smallint
    03 PIC_FMT char(30)
```

To allow a wide range of precision before and after the decimal point, the decimal number to be formatted is supplied as an integer (PIC_FMT_AMT) together with the number of decimals (PIC_FMT_SCALE).

This system component returns this output:

```
01 VRWFORMO
    03 RET_CODE smallint
    03 PIC_FMT_RESULT VarChar(30).
```

HPDTTM

This section presents the purpose, the syntax, and the usage of the HPDTTM component.

Purpose

This system component either provides the current date and time in a certain format or formats an integer number into a specified date and time template. Calls to this component are automatically invoked by the report program. You are not required to call this component in your driver rules.

Syntax

Here is the syntax of HPDTTM component.

HPDTTM component syntax

Name	HPDTTM
System Identifier	`HPDTTM`
Input View Name	`VRWDTTMI`
Input View Field	`DATE_FMT character(20)` Stores date format.
Input View Field	`TIME_FMT character(20)` Stores time format.
Input View Field	`FORMAT_IND character(1)` `Y` if formatting input data, `N` if formatting current date or time.
Input View Field	`INP_DATE_TIME picture('999999999')` Contains data for formatting (data must be numeric). Date is expected to be yyyymmdd. Time is expected to be hhmmsstt. If less than nine digits are passed, the component fills in leading zeroes.
Output View Name	`VRWDTTMO`
SubView Name	`VDTTM`
Output View Field	`DATE_STR character(18)` Formatted date.
Output View Field	`TIME_STR character(13)` Formatted time.
Output View Field	`YEAR_NBR integer(15)`
Output View Field	`MONTH_NBR integer(15)`
Output View Field	`DAY_NBR integer(15)`
Output View Field	`HOUR_NBR integer(15)`
Output View Field	`MIN_NBR integer(15)`
Output View Field	`SEC_NBR integer(15)`
Output View Field	`DAY_IN_YEAR integer(15)`
Output View Field	`DAY_PLUS_YEAR integer(31)`
Output View Field	`STATUS_CD picture('999999999')`

Obtaining the Current Date

To obtain the current date, the requirements shown in Requirements for current date field must be met.

Requirements for current date field

Field	Requirement
DATE_FMT	Desired date format must be specified

TIME_FMT	Must be cleared with the CLEAR statement
FORMAT_IND	Must equal N

Obtaining the Current Time

To obtain the current time, the requirements shown in Requirements for current time field must be met.

Requirements for current time field

Field	Requirement
DATE_FMT	Must be cleared with the CLEAR statement
TIME_FMT	Desired time format must be specified
FORMAT_IND	Must equal N

Formatting a Number into a Date Template

To format a number into one of the date templates, the requirements shown in Requirements for formatting a number into a date template must be met.

Requirements for formatting a number into a date template

Field	Requirement
DATE_FMT	Desired date format must be specified
TIME_FMT	Must be cleared with the CLEAR statement
FORMAT_IND	Must equal Y
INP_DATE_TIME	Number for formatting (for example 19920301)
For DATE_FMT=DATE1	DATE_STR returns 03/01/1992

Formatting a Number into a Time Template

To format a number into one of the date templates, the requirements shown in Requirements for formatting a number into a time template must be met.

Requirements for formatting a number into a time template

Field	Requirement
DATE_FMT	Must be cleared with the CLEAR statement
TIME_FMT	Desired time format must be specified
FORMAT_IND	Must equal Y
INP_DATE_TIME	Number for formatting (for example 000012345)
For TIME_FMT=TIME1	TIME_STR contains 00:00:12

You can simultaneously obtain both the current date and current time by providing appropriate non-blank entries for both DATE_FMT and TIME_FMT. But you cannot at the same time format a number into both a date and a time template.

STATUS_CODE Return Codes

The STATUS_CODE field for the HPDTTM component provides the return codes listed in Return codes for STATUS_CODE field.

Return codes for STATUS_CODE field

Return Code	Explanation
0	No error detected
1001	Invalid date format input

1002	Invalid time format input
1003	Cannot specify both date AND time request
1100	Invalid date
1200	Invalid time

HPSETVP

This section presents the purpose, the syntax, and the usage of the HPSETVP component.

Purpose

This system component is used in Report Writer in conjunction with (as input to) CDYNPIC to help generate dynamic pictures. You must make explicit calls to this component in your driver rules.

Syntax

Here is the syntax of HPSETVP component.
HPSETVP component syntax

Name	**HPSETVP**
System Identifier	`HPSETVP`
Input View Name	`HPSETVPI`
Input View Field	`KEY_LONG_NM character(30)` This is the report name.
Input View Field	`VPTAG character(16)`
Input View Field	`VPICTURE character(30)` This is the picture format. VPICTURE contains the picture specification you want to impose. If VPSCALE = 1, PIC_FMT_SCALE of VFORMATI is 1. Report Writer uses this to communicate the specific format in which the data is to be displayed.
Input View Field	`VPSCALE integer(15)`
Output View Name	`HPSETVPO`
Output View Field	`COMPLETE integer(31)`

System Information Components

Two components are available for retrieving information about the system. These components are:

- HPSMODE - returns the current execution environment
- HPSPARM - retrieves any parameters passed to the rule in the JCL that invoked the application

Because these components retrieve variables from the environment and do not process any input data, they have no input views.

HPSMODE

This section presents the purpose, the syntax, and the usage of the HPSMODE component.

Purpose

Because you can prepare a rule for different execution environments, you may wish to determine the current execution environment at runtime so you can perform certain environment-specific functions. This component returns the execution environment in which the rule that invoked the component is currently executing.

Syntax

Here is the syntax of HPSMODE component.
HPSMODE component syntax

Name	**HPSMODE**
System Identifier	`HPSMODE`
Input View Name	This component has no input view.
Output View Name	`HPSMODEO`
Output View Field	`EXECMODE character(6)` This field has the execution environment with these possible values: • CICS • IMS • BATCH • UNKNOWN

HPSPARM

This section presents the purpose, the syntax, and the usage of the HPSPARM component. See Example - Using HPSMODE and HPSPARM for an usage example.

Purpose

This component reads the EXEC CARD PARM statement in the JCL that invoked the application and returns that parameter to the calling rule.

LE statements in the PARM statement is ignored.

Syntax

Here is the syntax of HPSPARM component.
HPSPARM component syntax

Name	**HPSPARM**
System Identifier	`HPSPARM`
Input View Name	This component has no input view.
Output View Name	`HPSPARMO`
Output View Field	`EXECPARM character(78)` The parameter passed to the application in the EXEC CARD PARM statement in the JCL

Usage

If the application was not invoked with JCL (for example, it was invoked using TE), the EXECPARM field contains the value "PARAMETER". If you are testing with TE, you need to break on this component and set the EXECPARM field correctly.

Example - Using HPSMODE and HPSPARM

The following sample rule, HPSIR01, illustrates the use of these components. This rule uses HPSPARM to find the EXEC card parameter and HPSMODE to find the execution environment. It then uses HPWRITE to write this information to a data set.

```
*>**************************************************\**<*
*> HPSIR01 <*
*>*\**************************************************\**<*
map 'FILE' TO DDNAME OF HPOPENI
map 'OUTPUT' TO MODE OF HPOPENI
use component HPOPEN
if COMPLETE OF HPOPENO >0
    RETURN
map DCB OF HPOPENO TO DCB OF HPWRITEI
use component HPSPARM
map EXECPARM OF HPSPARMO TO RECORD OF HPWRITEI
map 80 TO SIZE OF HPWRITEI
use component HPWRITE
use component HPSMODE
map EXECMODE OF HPSMODEO TO RECORD OF HPWRITEI
map 80 TO SIZE OF HPWRITEI
use component HPWRITE
map DCB OF HPWRITEI TO DCB OF HPCLOSEI
use component HPCLOSE
```

The JCL for this rule is included. (The rule's system identifier is "AAAUVV".)

```
//SS0116C JOB (NA2,HPSD),'RUN PROG',CLASS=1,
//       NOTIFY=SS0116,MSGCLASS=X,REGION=4096K
//RUN      EXEC PGM=HPSBATCH,PARM='AAAUVV,,X,Y,Z,1'
//STEPLIB  DD DSN=SS0196.HPS510.LOADLIB,DISP=SHR
//         DD DSN=SS0196.USER1.BATCH.LOADLIB,DISP=SHR
//         DD DSN=MV.COBOLII.COB2LIB,DISP=SHR
//         DD DSN=MV.PLI2.SIBMLINK,DISP=SHR
//         DD DSN=MV.EDC.SEDCBASE,DISP=SHR
//         DD DSN=MV.EDC.SIBMBASE,DISP=SHR
//         DD DSN=MV.EDC.SEDCLINK,DISP=SHR
//         DD DSN=MV.EDC.SIBMLINK,DISP=SHR
//         DD DSN=MV.EDC.SEDCCOMP,DISP=SHR
//         DD DSN=MV.EDC.SEDCMSGS,DISP=SHR
//         DD DSN=MV.EDC.SEDCHDRS,DISP=SHR
//INPUT    DD DUMMY
//OUTPUT   DD SYSOUT=\*
//VIEWFILE DD DSN=NAVHP.SS0196.USER1.BATCH.VIEWDEF,DISP=SHR
//FILE1 DD DSN=SS0116.HPS510.T10200.FILE1,DISP=SHR
//SYSTSPRT DD SYSOUT=\*
//SYSABOUT DD SYSOUT=\*
//SYSPRINT DD SYSOUT=\*
//SYSUDUMP DD SYSOUT=\*
//CEEDUMP  DD SYSOUT=\*
//PLIDUMP  DD SYSOUT=\*
/*
```

OpenCOBOL for Unix and Java Batch System Components

A number of components are available in the repository for execution in the OpenCOBOL for Unix & Java batch environments:

- Input and Output Components - for handling QSAM input and output of fixed and variable length records
- System Information Components - for retrieving system information

The input and output components deal with the related issue of maintaining positioning within sequential files. The functions they perform include opening, reading, writing, and closing files.
The system information components retrieve variables from the environment and do not process any input data.
You communicate with these components using standard input and output view passing. By convention, a component's input view is the name of the component with the letter "I" appended. Similarly, the name of its output view is the component's name with the letter "O" appended.

Input and Output Components

These components maintain sequential files and positioning within them. The managed files can be fixed length or variable length and blocked or unblocked, but you must define them before referencing them. The functions they perform include opening, reading, writing, and closing sequential files. In addition, the SET_HPSBATCH_RETURN_CODE component allows you to set the value of the return code that the HPSBATCH program returns.
This set of components performs the same functionality as the set of similar components in Mainframe Batch System Components section, but uses the LOCATE I/O mode. This mode passes pointers to data, instead of the data themselves, and hence is more efficient:

- HPS_OPEN_FILE_LOCATE_MODE (opens a sequential file)
- HPS_READ_FILE_LOCATE_MODE (reads from a sequential file)
- HPS_WRITE_FILE_LOCATE_MODE (writes to a sequential file)
- HPS_CLOSE_FILE_LOCATE_MODE (closes a sequential file)

There is also a component for flushing data records stored in I/O buffers:

- HPS_TRUNC_FILE_LOCATE_MODE

There is also a component that sets the value of the return code that the HPSBATCH program returns:

- SET_HPSBATCH_RETURN_CODE

HPS_OPEN_FILE_LOCATE_MODE

This section presents the purpose, the syntax, and the usage of the HPS_OPEN_FILE_LOCATE_MODE component.

Purpose

This component initiates processing a sequential mainframe data set through the standard QSAM interface. Storage for a mainframe data set control block (DCB) is allocated by this component. The DCB is then initialized and opened for either input or output depending on the value of the MODE field.
This component is functionally equivalent to HPOPEN, except that this component uses LOCATE I/O mode instead of MOVE I/O mode, and hence is more efficient.

Syntax

Here is the syntax of HPS_OPEN_FILE_LOCATE_MODE component.

HPS_OPEN_FILE_LOCATE_MODE component syntax

Name	**HPS_OPEN_FILE_LOCATE_MODE**
System Identifier	CHPOPEN
Input View Name	HPS_OPEN_FILE_LOCATE_MODE_I
Input View Field	DDNAME character(8) Identifies the JCL file in a corresponding TSO ALLOCATE command or DD card
Input View Field	MODE character(8) Can be one of two values: • INPUT indicates data is taken from the data set (default) • OUTPUT indicates data is passed to the data set
Output View Name	HPS_OPEN_FILE_LOCATE_MODE_O
Output View Field	COMPLETE integer(31) The following codes are returned on the workstation: • 0 = Successful • 4 = File could not be opened or invalid mode parameter The following codes are returned on the mainframe: • 0 = Successful • 4 = File could not be opened
Output View Field	LOCATE_MODE_DCB character(4) Address of the open data set control block (DCB).

Usage

Missing DD cards for the files concerned are the most common cause of open failures. Do not try to process a file that does not open successfully. In the case of an open failure, the DCB contents are meaningless.

OpenCOBOL for Unix & Java Batch Support

This component is supported for generation of OpenCOBOL for Unix & Java Batch applications. The error codes are the same as those for the mainframe. This component reads the INI file and environment variables for a given DDNAME (see DDNAME in OpenCOBOL for Unix & Java Batch), then opens the file as binary according to the MODE:

- **INPUT** - The file is opened for reading.
- **OUTPUT** - The file is opened for writing.
- **APPEND** - The file is opened for appending.
- **UPDATE** - The file is opened for update (read and write).

Even in the Line Sequential mode, the file is opened as binary; Line Feed and Carriage Return characters are processed by components. See Line Sequential Record Format.

DDNAME in OpenCOBOL for Unix & Java Batch

HPS_OPEN_FILE_LOCATE_MODE component receives DDNAME as an argument. For OpenCOBOL for Unix & Java Batch, it must be translated to a real filename using the information from the environment variables and the INI file. The INI file is set up in the environment variable FILEINI. When invoking the root rule directly, you must set up this variable.
The algorithm of reading the settings for DDNAME is as follows:

1. Open the INI file, and search for the requested DDNAME record in the following format:

   ```
   <DDNAME>=[["]<FILE_NAME>["]][,RECFM={V|F|L}]\[,LRECL=<record length>]
   ```

   where:

   - FILE_NAME is a real file name.
   - RECFM is a record format: **V**ariable, **F**ixed or **L**ine sequential (or text mode)
   - LRECL is a record length.
2. Read the environment variables named <DDNAME> which must be in the same format:

   ```
   [["]<FILE_NAME>["]][,RECFM={V|F|L}][,LRECL=<record length>]
   ```

Settings in the environment variables override the corresponding settings read from the INI file. Note the following for RECFM and LRECL settings:

- If RECFM is not specified, RECFM=V is assumed.
- If RECFM=F, then LRECL is mandatory and must be greater than zero.
- If RECFM=V or RECFM=L, then LRECL is optional.

HPS_READ_FILE_LOCATE_MODE

This section presents the purpose, the syntax, and the usage of the HPS_READ_FILE_LOCATE_MODE component.

Purpose

This component reads records from a sequential data set through the standard QSAM interface. It executes a GET to read a single record from the input file. The record data is then copied from an internal buffer below the 16-megabyte line to view specified by the LOCATE_RECORD field. This component is functionally equivalent to HPREAD, except that this component uses LOCATE I/O mode instead of MOVE I/O mode, and hence is more efficient.

Syntax

Here is the syntax of HPS_READ_FILE_LOCATE_MODE component.

HPS_READ_FILE_LOCATE_MODE component syntax

Name	**HPS_READ_FILE_LOCATE_MODE**
System Identifier	CHPREAD
Input View Name	HPS_READ_FILE_LOCATE_MODE_I
Input View Field	VIEW_LONG_NAME character(30) The name of the view to be read

Input View Field	SIZE_LONG integer(31) Record length in bytes
Input View Field	LOCATE_MODE_DCB character(4) Address of the data set control block (DCB) obtained through use of the HPS_OPEN_FILE_LOCATE_MODE component
Input View Field	LOCATE_RECORD character(8) Location of the view structure where the component should place the data after it has read them from the record. Use the LOC function to supply the view location. (Refer to the *Rules Language Reference Guide*.)
Output View Name	HPS_READ_FILE_LOCATE_MODE_O
Output View Field	COMPLETE integer(31) The following codes are returned on the workstation: • 0 = Successful • 4 = End of file reached • 8 = Invalid DCB or the read operation failed The following codes are returned on the mainframe: • 0 = Successful, 4 = End of file reached.
Output View Field	SIZE_LONG integer(31) Record length

Usage

It is imperative that the value supplied to the SIZE_LONG field in the component input view not exceed the size of the view into which the record is being read. It is recommended that the function SIZEOF(viewname) be used to determine the value to supply to the component. For example:

```
map SIZEOF(viewname) to SIZE_LONG of HPS_READ_FILE_LOCATE_MODE_I
```

OpenCOBOL for Unix & Java Batch Support

This component is supported for generation of OpenCOBOL for OpenCOBOL for Unix & Java Batch. The error codes are the same as those for the mainframe. LOCATE_RECORD of the input view is initialized with SIZE_LONG of the input view number of spaces. It reads records according to the record format which is described in the DDNAME in OpenCOBOL for Unix & Java Batch:

- **Fixed Record Format**
  If SIZE_LONG of the input view is zero, then LRECL is used to determine the number of bytes to read. If SIZE_LONG of the input view is not zero and is less than LRECL, then an error code is returned and nothing is read. The number of bytes specified in LRECL is read from LOCATE_RECORD. If the number of bytes read is less than the record lengths, no error is returned. SIZE_LONG of the output view always contains the actual number of bytes read.
- **Variable Record Format**
  It reads 4 bytes from the file and subtracts four from an integer value formed by the first 2 bytes, and assigns an integer value formed by all 4 bytes to SIZE_LONG of the output view. This is the record length. Then it reads the number of bytes specified in SIZE_LONG of the output view into LOCATE_RECORD. SIZE_LONG of the input view and LRECL are ignored.
- **Line Sequential Record Format**
  First, the expected maximum length of the record is determined: if SIZE_LONG of the input view is greater than zero, then it is used as the buffer length; otherwise, LRECL is used. If both SIZE_LONG of the input view and LRECL are less than or equal to zero, then an error code is returned and nothing is read. LOCATE_RECORD is initialized with spaces by the determined maximum record length. The file is read into LOCATE_RECORD until either the buffer is full, or newline or EOF is encountered. SIZE_LONG of the output view contains the number of bytes actually read, not counting the newline character. If the buffer becomes full and neither newline nor EOF has been read, then the remainder of the record is skipped and lost.

HPS_WRITE_FILE_LOCATE_MODE

This section presents the purpose, the syntax, and the usage of the HPS_WRITE_FILE_LOCATE_MODE component.

Purpose

This component writes records to a sequential mainframe data set through the standard QSAM interface. It copies the record data specified by the LOCATE_RECORD field of the input view to an internal buffer below the 16-megabyte line. A PUT is then executed to write the single record to the output file.
This component is functionally equivalent to HPWRITE, except that this component uses LOCATE I/O mode instead of MOVE I/O mode, and therefore is more efficient.

Syntax

Here is the syntax of HPS_WRITE_FILE_LOCATE_MODE component.

HPS_WRITE_FILE_LOCATE_MODE component syntax

Name	**HPS_WRITE_FILE_LOCATE_MODE**
System Identifier	CHPWRIT
Input View Name	HPS_WRITE_FILE_LOCATE_MODE_I
Input View Field	VIEW_LONG_NAME character(30) Name of the view to be written
Input View Field	SIZE_LONG integer(31) Record length in bytes
Input View Field	LOCATE_MODE_DCB character(4) Address of the data set control block (DCB) obtained through use of the HPS_OPEN_FILE_LOCATE_MODE component
Input View Field	LOCATE_RECORD character(8) Location of the view structure where the component gets the data for writing to the record. Use the LOC function to supply the location of a view. (Refer to the *Rules Language Reference Guide* .)
Output View Name	HPS_WRITE_FILE_LOCATE_MODE_O
Output View Field	COMPLETE integer(31) The following codes are returned on the workstation: • 0 = Success • 8 = Invalid DCB or the write operation failed The following codes are returned on the mainframe: • 0 = Success • 4 = Could not be written

Usage

The SIZE_LONG field must contain the length of the record to be written when processing variable-length records. It should not include the length of a record descriptor word. The contents of the SIZE_LONG field are ignored when processing fixed-length records. (The DCB contains the necessary record length information.)

OpenCOBOL for Unix & Java Batch Support

This component is supported for generation of OpenCOBOL for OpenCOBOL for Unix & Java Batch. The error codes are the same as those for the mainframe. It writes records to the output file according to the record format, which is described in the DDNAME in OpenCOBOL for Unix & Java Batch:

- **Fixed Record Format**
  First, the number of bytes to be taken from LOCATE_REOCRD is determined. If SIZE_LONG of the input view is zero, then LRECL is used; otherwise, SIZE_LONG is used. Next, this number of bytes is taken from LOCATE_RECORD of the input view, and written to the output file.
- **Variable Record Format**
  When the record format is the variable format, it writes SIZE_LONG of the input view plus 4 (total length of this record) to the file, placing it into 2 bytes, and writes two 0 bytes (this is for mainframe component compatibility). Then the number of bytes specified in SIZE_LONG is taken from LOCATE_RECORD, and written to the file. For the variable record format, LRECL is ignored.
- **Line Sequential Record Format**
  The number of bytes specified in SIZE_LONG is taken from LOCATE_RECORD and written to the file. Then the newline character (\n) is written. For the line sequential record format, LRECL is ignored.

HPS_CLOSE_FILE_LOCATE_MODE

This section presents the purpose, the syntax, and the usage of the HPS_CLOSE_FILE_LOCATE_MODE component.

Purpose

This component concludes processing a sequential mainframe data set through the standard QSAM interface. It closes the file, causing to take the appropriate actions for the data set, as specified in the JCL. Storage for the data set control block (DCB) and the file buffer pools is freed by this component.
This component is functionally equivalent to HPCLOSE, except that this component uses LOCATE I/O mode instead of MOVE I/O mode, and hence is more efficient.

Syntax

Here is the syntax of HPS_CLOSE_FILE_LOCATE_MODE component.

HPS_CLOSE_FILE_LOCATE_MODE component syntax

Name	**HPS_CLOSE_FILE_LOCATE_MODE**
System Identifier	CHPCLOS
Input View Name	HPS_CLOSE_FILE_LOCATE_MODE_I
Input View Field	LOCATE_MODE_DCB character(4) Address of the DCB obtained through the use of the HPS_OPEN_FILE_LOCATE_MODE component
Output View Name	HPS_CLOSE_FILE_LOCATE_MODE_O
Output View Field	COMPLETE integer(31) The following codes are returned on the workstation: • 0 = Success • 4 = Invalid DCB or the close operation failed The following code is returned on the mainframe: • 0 = Success This is the only value returned. Any condition causing a close to fail usually causes the batch job to abend (for example, a missing DD statement).

Usage

Do not try to process a closed file; first open it.

OpenCOBOL for Unix & Java Batch Support

This component is supported for generation of OpenCOBOL for OpenCOBOL for Unix & Java Batch. The error codes are the same as those for the mainframe.

HPS_TRUNC_FILE_LOCATE_MODE

This section presents the purpose, the syntax, and the usage of the HPS_TRUNC_FILE_LOCATE_MODE component.

Purpose

Batch jobs that use the mainframe checkpoint/restart feature to recover from failure can experience a loss of data from files used by rules. If you lose data in this way, you can use this system component to flush data records stored in the I/O buffers. This component issues the mainframe TRUNC macro command for the specified file, which causes the operating system to regard the current buffer as full, allowing a short block to be written to the file.

Syntax

Here is the syntax of HPS_TRUNC_FILE_LOCATE_MODE component.

HPS_TRUNC_FILE_LOCATE_MODE component syntax

Name	**HPS_TRUNC_FILE_LOCATE_MODE**
System Identifier	CHPTRUN
Input View Name	HPS_TRUNC_FILE_LOCATE_MODE_I
Input View Field	LOCATE_MODE_DCB character(4) Address of the DCB obtained through the use of the HPS_OPEN_FILE_LOCATE_MODE component
Output View Name	HPS_TRUNC_FILE_LOCATE_MODE_O

Output View Field	COMPLETE integer(31) The following codes are returned on the workstation: • 0 = Success • 4 = Invalid DCB or the close operation failed The following code is returned on the mainframe: • 0 = Success This is the only value returned. Any condition causing a close to fail usually causes the batch job to abend (for example, a missing DD statement).

OpenCOBOL for Unix & Java Batch Support

This component is supported for generation of OpenCOBOL for OpenCOBOL for Unix & Java Batch. It flushes all data that has not been written to the file if a file was opened for writing. If a file was opened for reading, it clears content of the internal read buffer. The error codes are the same as those for the mainframe.

SET_HPSBATCH_RETURN_CODE

This section presents the purpose, the syntax, and the usage of the SET_HPSBATCH_RETURN_CODE component.

Purpose

This component sets the value of the return code that the HPSBATCH program returns. Executing this component twice within the same rule displays only the last value assigned.

Syntax

Here is the syntax of SET_HPSBATCH_RETURN_CODE component.

SET_HPSBATCH_RETURN_CODE component syntax

Name	SET_HPSBATCH_RETURN_CODE
System Identifier	HPBRETC
Input View Name	SET_HPSBATCH_RETURN_CODE_I
Input View Field	RETURN_CODE integer(15) This can be a number from 0 to 4,095

OpenCOBOL for Unix & Java Batch Support

This component is supported for generation of OpenCOBOL for OpenCOBOL for Unix and Java Batch. The error codes are the same as those for the mainframe.
When rule execution ends, the value passed in the RETURN_CODE field is stored and assigned to the COBOL register RETURN-CODE only. A call to this component does not affect the RETURN-CODE register while the rule executes. The value passed to the last executed call of this component is used as RETURN-CODE.

System Information Components

Two components are available for retrieving information about the system. These components are:

- HPSMODE - returns the current execution environment
- HPSPARM - retrieves any parameters passed to the rule in the JCL that invoked the application

Because these components retrieve variables from the environment and do not process any input data, they have no input views.

HPSMODE

This section presents the purpose, the syntax, and the usage of the HPSMODE component.

Purpose

Because you can prepare a rule for different execution environments, you may wish to determine the current execution environment at runtime so you can perform certain environment-specific functions. This component returns the execution environment in which the rule that invoked the component is currently executing.

Syntax

Here is the syntax of HPSMODE component.

HPSMODE component syntax

Name	**HPSMODE**
System Identifier	HPSMODE
Input View Name	This component has no input view.
Output View Name	HPSMODEO
Output View Field	EXECMODE character(6) This field has the execution environment with these possible values: • CICS • IMS • BATCH • UNKNOWN

OpenCOBOL for Unix & Java Batch Support

This component is supported for generation of OpenCOBOL for OpenCOBOL for Unix and Java Batch. It sets EXECMODE of the output view to "BATCH" upon return.

HPSPARM

This section presents the purpose, the syntax, and the usage of the HPSPARM component.

Purpose

This component reads the EXEC CARD PARM statement in the JCL that invoked the application and returns that parameter to the calling rule. LE statements in the PARM statement is ignored.

Syntax

Here is the syntax of HPSPARM component.

HPSPARM component syntax

Name	**HPSPARM**
System Identifier	HPSPARM
Input View Name	This component has no input view.
Output View Name	HPSPARMO
Output View Field	EXECPARM character(78) The parameter passed to the application in the EXEC CARD PARM statement in the JCL

Usage

If the application was not invoked with JCL (for example, it was invoked using TE), the EXECPARM field contains the value "PARAMETER". If you are testing with TE, you need to break on this component and set the EXECPARM field correctly.

OpenCOBOL for Unix & Java Batch Support

This component is supported for generation of OpenCOBOL for OpenCOBOL for Unix & Java Batch. It reads the HPSPARM environment variable and returns it in the EXECPARM field of the output view.

IMS Components

There are three system components that you use when building IMS applications for the mainframe:

- GET_IMS_PCB_ADDR
- IMS_SPA_COMPONENT

- STOP_TP_RULE_REGION

GET_IMS_PCB_ADDR

This section presents the purpose, the syntax, and the usage of the GET_IMS_PCB_ADDR component. See Example: GET_IMS_PCB_ADDR for an usage example.

Purpose

Use this component to obtain the address of the Program Control Block (PCB). You can then pass this address to a user-written component to allow it to access an IMS database in the conventional (non-AIB) way.

Syntax

Here is the syntax of GET_IMS_PCB_ADDR component.

GET_IMS_PCB_ADDR component syntax

Name	`GET_IMS_PCB_ADDR`
System Identifier	`HPIGPCB`
Input/Output View Name	`GET_IMS_PCB_ADDR_VIEW`
View Field	`IMS_PCB_LIST_ADDR integer(31)`

Example: GET_IMS_PCB_ADDR

This example shows the usage of this component.

```
use COMPONENT GET_IMS_PCB_ADDR
```

Pass the IMS_PCB_LIST_ADDR field to the user component that accesses IMS databases in the conventional way. The user component can now reference that field as an address and base the list of PCBs to it.

```
map IMS_PCB_LIST_ADDR of GET_IMS_PCB_ADDR_VIEW to PCB_BASE_ADDRESS of USER_COMPONENT_I
use COMPONENT USER_COMPONENT
```

⚠ GET_IMS_PCB_ADDR is not supported under OpenCOBOL execution.

IMS_SPA_COMPONENT

This section presents the purpose, the syntax, and the usage of the IMS_SPA_COMPONENT component. See Example: IMS_SPA_COMPONENT for an usage example.

Purpose

Use this component to allow rules with an execution environment of IMS to manipulate the IMS Scratch Pad Area (SPA) for IMS conversational transactions. This component supports the following functions.

Functions supported by IMS_SPA_COMPONENT

Retrieve SPA	To enable a rule to view the SPA, map R to SPA_FUNCTION_CODE before calling this component. The output view of this component contains the SPA_DATA or a non-blank SPA_RETURN_CODE code.
Update SPA	To update the SPA, map U to SPA_FUNCTION_CODE and map data to the SPA_DATA. The output view of this component contains the updated SPA_DATA or a non-blank SPA_RETURN_CODE code.
Insert SPA	To combine the Update SPA_FUNCTION_CODE and an IMS SPA Insert. It sends the SPA back to IMS with a new non-AppBuilder transaction code. You can use this function only for an immediate conversational transaction switch from an AppBuilder rule to a non-AppBuilder component or from one non-AppBuilder component to another. Map I to the SPA_FUNCTION_CODE to inform this component to insert the SPA. You must always insert an SPA before inserting any other messages, as this component uses the SPA to modify the non-Express Alternate PCB to specify the new conversational message queue.

Insert Messages	After inserting the SPA, you must insert a further message to complement the SPA call. Map M to the SPA_FUNCTION_CODE to insert the message.

Syntax

Here is the syntax of IMS_SPA_COMPONENT component.

IMS_SPA_COMPONENT component syntax

Name	`IMS_SPA_COMPONENT`
System Identifier	`HPISPAU`
Input View Name	`SPA_INPUT_VIEW`
Input View Field	`SPA_FUNCTION_CODE character(1)` Values are: R = Retrieve SPA, U = Update SPA, I = Insert SPA, M = Insert Message
Input View Field	`SPA_FILLER_BYTE1 character(1)`
Input View Field	`SPA_TRAN_CODE character(8)`
Input View Field	`SPA_LENGTH integer(15)` Do not use this field for an Update or an Insert SPA function. (The size in the LL bytes of the original SPA is used and is not changed.) You must use this field for an Insert Message function.
Input View Field	`SPA_DATA character(1986)` Use the first 8 bytes to specify the IMS transaction code if the function is an Insert SPA or Update SPA. This field does not include the IMS LL and ZZ bytes. They are calculated by the component.
Output View Name	`SPA_OUTPUT_VIEW`
Output View Field	`SPA_RETURN_CODE character(2)` • 01 = Invalid SPA function code • 02 = SPA address does not exist (hex zeroes) • 03 = SPA input view is not large enough to contain the SPA • A1 = Call attempted to unknown transaction • AT = User's message area too long • A6 = Message segment size exceeded • A7 = Number of output messages inserted has been exceeded • XB = SPA already inserted • XG = Attempted to insert SPA with different or non-fixed length • X2 = The SPA must be inserted first • X4 = The non-AppBuilder transaction is not conversational; Insert SPA not allowed • X5 = SPA already inserted • X6 = Invalid transaction code in SPA
Output View Field	`SPA_DATA character(1986)`

Usage

If you are doing an immediate conversational transfer to a non-AppBuilder application, and the current transaction SPA is defined as fixed length, then the new transaction's SPA also must be defined as fixed length. A variable length SPA for the current transaction can transfer to both fixed and variable length SPA conversations. The SPA length is not changed by the component. Refer to your IMS operating system manuals for full details.

To allow a rule to reuse the input view of this component for both an SPA Insert and a Message Insert, use the OVERLAY verb to populate the data area. See the sample rule above.

The SPA must contain the new IMS transaction code that this component transfers to in the first 8 bytes of the SPA_data area.

The SPA size does not change for the Update or Insert SPA functions.

The maximum size for an SPA is 32,000 bytes.

Example: IMS_SPA_COMPONENT

This example performs an immediate conversational transfer to a non-AppBuilder user application using this component.

The following internal declaration is used for the second call to the component, because the input view for the Insert Message is different from the

Insert SPA.

```
dcl
   spa_function_code char (1);
   spa_length        fixed bin (15);
   last_name         char (15);
   parm_func         char (1);

   spa_message_view VIEW CONTAINS
   spa_function_code,
   spa_length,
   last_name,
   parm_func;
enddcl
```

Map the transaction code from the rule's input view to the input view of the component.

```
map SPA_TRANSACTION_CODE of SPA_INPUT_UPD_VIEW to SPA_TRAN_CODE of SPA_INPUT_VIEW
```

Assign the Insert SPA function.

```
map 'I' to SPA_FUNCTION_CODE of SPA_INPUT_VIEW
```

Call the component.

```
use component IMS_SPA_COMPONENT
```

Assign the Insert Message function to the input view of the component.

```
map 'M' to SPA_FUNCTION_CODE of SPA_MESSAGE_VIEW
map LAST_NAME of SPA_INPUT_UPD_VIEW to LAST_NAME of SPA_MESSAGE_VIEW
map PARM_FUNC of SPA_INPUT_UPD_VIEW to PARM_FUNC of SPA_MESSAGE_VIEW
```

Calculate the length of the message area without the IMS LL/ZZ bytes.

```
map 18 to SPA_LENGTH of SPA_MESSAGE_VIEW
```

The OVERLAY verb allows the internally declared structure to be mapped to the SPA_INPUT_VIEW of the component, allowing for the same rule to call the component with two different input views.

```
overlay SPA_MESSAGE_VIEW to SPA_INPUT_VIEW
use component IMS_SPA_COMPONENT
```

STOP_TP_RULE_REGION

This section presents the purpose, the syntax, and the usage of the STOP_TP_RULE_REGION component.

Purpose

Use this component to notify the IMS Application Execution Run Control subsystem that an independent IMS TP region for the specified application is to be stopped.

Syntax

Here is the syntax of STOP_TP_RULE_REGION component.

STOP_TP_RULE_REGION component syntax

Name	`STOP_TP_RULE_REGION`
System Identifier	`HPISTPR`
Input/Output View Name	`RULE_STOP_INIT`
View Field	`RETURN_KEY character(2)` Contains any invalid IMS return code
View Field	`IMS_RULE_ID character(8)` The rule's system identifier

Example: STOP_TP_RULE_REGION

This example initiates another TP rule 100 times, passing it different input view field values on each call.

```
do while (I <= 100)
   use RULE SAMPLE_RULE INIT
   map I + 1 to I
enddo
```

After the rule has been called for the last time, this rule or another rule issues a call to the component to request the completion of the process.

```
map AAOVWX to IMS_RULE_ID of RULE_STOP_INIT
use COMPONENT STOP_TP_RULE_REGION
```

Check the RETURN_KEY field; any invalid IMS return code detected by the component is put into this field.

BLOB System Components

Some system components allow you to manipulate large-object files. Large-object files can be either text files or binary files. In this topic, such files are referred to generically as BLOBs. You can use these components only on the mainframe host. These components with the Rules Language can accomplish the BLOB Functions.

The following is a list of BLOB components with a description of how each component manipulates large object files:

List of BLOB components

HPS_BLOB_CLOSE_FILE	Closes a BLOB file that was previously opened with the HPS_BLOB_OPEN_FILE component.
HPS_BLOB_GENNAME_FILE	Allocates a BLOB file, which is a VSAM file to which you can write BLOB contents.
HPS_BLOB_OPEN_FILE	Opens a BLOB file. The file can be created on the mainframe with the HPS_BLOB_GENNAME_FILE component, or it can be created automatically when the file was transferred to the mainframe from another system.
HPS_BLOB_READ_FILE	Performs a sequential read of a BLOB file.
HPS_BLOB_WRITE_FILE	Appends to a BLOB file.
HPS_BLOB_DELETE_FILE	Deletes a BLOB file.

BLOB Functions

The BLOB components with the Rules Language can accomplish the following functions:

- Transferring BLOB Files from Workstation to Mainframe and accessing them on the mainframe
- Creating a BLOB file on the Mainframe
- Transferring BLOB Files from Mainframe to Workstation

Transferring BLOB Files from Workstation to Mainframe

To transfer a BLOB file residing on a workstation to the mainframe and to access it on the mainframe, do the following:

1. In a workstation rule, map the BLOB file name to a TEXT or IMAGE field in the input view of a mainframe rule.
   The fact that a TEXT or IMAGE field contains a file name in the input view of a mainframe rule tells the system that the workstation is sending a BLOB file to the mainframe. A TEXT field indicates the file contains ASCII. An IMAGE field indicates the file is binary.
   When the workstation rule invokes the mainframe rule, the system transfers the BLOB file to the mainframe. The System overwrites the name of the workstation file with the name of the file that is automatically allocated on the mainframe.
2. On the mainframe, use the file name that the system overwrites in the TEXT or IMAGE field of the rule's input view as input to HPS_BLOB_OPEN_FILE to open the file.
3. Use the file handle returned by HPS_BLOB_OPEN_FILE as input to HPS_BLOB_READ_FILE to read the contents of the BLOB file.
4. Use the file handle as input to HPS_BLOB_CLOSE_FILE.
5. If the workstation rule wants to know the name of the file that was allocated on the mainframe, then the mainframe rule should map the TEXT or IMAGE field from its input view to its output view. This would allow a workstation rule to later download the file.

Creating a BLOB file on the Mainframe

To create a BLOB file on the mainframe, do the following:

1. Use HPS_BLOB_GENNAME_FILE to allocate a file.
2. Use the name returned by HPS_BLOB_GENNAME_FILE as input to HPS_BLOB_OPEN_FILE to open the file.
3. Use the file handle returned by HPS_BLOB_OPEN_FILE as input to HPS_BLOB_WRITE_FILE to write the BLOB contents to the file.
4. Use the file handle as input to HPS_BLOB_CLOSE_FILE.

Transferring BLOB Files from Mainframe to Workstation

To transfer a BLOB file from the mainframe to a workstation, do the following:

In a mainframe rule, map the BLOB file name to a TEXT or IMAGE field in the mainframe rule's output view:

- If the BLOB file was created on the mainframe, then the file name is the one returned by HPS_BLOB_GENNAME_FILE.
- If the BLOB file was uploaded from a workstation, then the file name is the one passed in a TEXT or IMAGE field in the input view of the mainframe rule when the file was uploaded.

The fact that a TEXT or IMAGE field contains a file name in the output view of a mainframe rule tells the system that the workstation is receiving a BLOB file from the mainframe. A TEXT field indicates the file contains EBCDIC. An IMAGE field indicates it is binary.

When the mainframe rule returns control to the workstation rule, the BLOB file is automatically transferred to the workstation. The workstation directory to which the file is transferred is specified in the FILE_XFER_LOCATION field of the dna.ini file.

⚠ Downloading from a mainframe to a workstation is initiated by a workstation rule. It is the workstation rule that uses the mainframe rule, and the mainframe rule that returns the file as its output.

HPS_BLOB_CLOSE_FILE

This section presents the purpose, the syntax, and the usage of the HPS_BLOB_CLOSE_FILE component.

Purpose

Use this component to close a previously opened BLOB file. The file can be created on the mainframe with the HPS_BLOB_GENNAME_FILE component, or it can be created automatically when the file was transferred to the mainframe from another system.

Syntax

Here is the syntax of HPS_BLOB_CLOSE_FILE component.

HPS_BLOB_CLOSE_FILE component syntax

Name	`HPS_BLOB_CLOSE_FILE`
System Identifier	`HPDCLOS`
Input View Name	`HPS_BLOB_CLOSE_FILE_I`
Input Field Name	`HPS_BLOB_FILE_HANDLE integer(31)` File handle returned by the HPS_BLOB_OPEN_FILE component in the HPS_BLOB_FILE_HANDLE field of its output view. After using this component, this file handle is no longer valid.

Output View Name	`HPS_BLOB_CLOSE_FILE_O`
Output Field Name	`COMPLETE integer(31)` 0 = Successful 1 = Unsuccessful

HPS_BLOB_DELETE_FILE

This section presents the purpose, the syntax, and the usage of the HPS_BLOB_DELETE_FILE component.

Purpose

Use this component to delete a previously allocated BLOB file. The file can be created on the mainframe with the HPS_BLOB_GENNAME_FILE component, or it can be created automatically when the file was transferred to the mainframe from another system.

Syntax

Here is the syntax of HPS_BLOB_DELETE_FILE component.

HPS_BLOB_DELETE_FILE component syntax

Name	**`HPS_BLOB_DELETE_FILE`**
System Identifier	`HPDDELE`
Input View Name	`HPS_BLOB_DELETE_FILE_I`
Input Field Name	`HPS_BLOB_FILE_HANDLE integer(31)` File handle returned by the HPS_BLOB_OPEN_FILE component in the HPS_BLOB_FILE_HANDLE field of its output view. After using this component, this file handle is no longer valid.
Output View Name	`HPS_BLOB_DELETE_FILE_O`
Output Field Name	`COMPLETE integer(31)` 0 = Successful 1 = Unsuccessful

HPS_BLOB_GENNAME_FILE

This section presents the purpose, the syntax, and the usage of the HPS_BLOB_GENNAME_FILE component.

Purpose

Use this component to allocate a VSAM file that you can use to store the contents of a BLOB file before transferring the file to another system.

Syntax

Here is the syntax of HPS_BLOB_GENNAME_FILE component.

HPS_BLOB_GENNAME_FILE component syntax

Name	**`HPS_BLOB_GENNAME_FILE`**
System Identifier	`HPDGENN`
Input View Name	`HPS_BLOB_GENNAME_FILE_I`
Input Field Name	`HPS_BLOB_LENGTH small integer or integer(31)` Length of field in which the generated file name is to be returned. You can use the SIZEOF function to calculate a length. The field must be at least 26 bytes.
Input Field Name	`HPS_BLOB_FILE_NAME character(8)` Location of a field in which the generated file name is to be returned. Use the LOC function to determine a location. (Refer to the *Rules Language Reference Guide*.) Because the LOC function takes a view as its argument, the field must be the first field in a view.
Output View Name	`HPS_BLOB_GENNAME_FILE_O`

Output Field Name	`COMPLETE small integer or integer(31)` 0 = Successful 1 = Unsuccessful

HPS_BLOB_OPEN_FILE

This section presents the purpose, the syntax, and the usage of the HPS_BLOB_OPEN_FILE component.

Purpose

Use this component to open a previously allocated BLOB file. The file can be created on the mainframe with the HPS_BLOB_GENNAME_FILE component, or it can be created automatically when the file was transferred to the mainframe from another system.

Syntax

Here is the syntax of HPS_BLOB_OPEN_FILE component.

HPS_BLOB_OPEN_FILE component syntax

Name	**`HPS_BLOB_OPEN_FILE`**
System Identifier	`HPDOPEN`
Input View Name	`HPS_BLOB_OPEN_FILE_I`
Input Field Name	`HPS_BLOB_FILE_MODE character(8)` Can be one of two values. INPUT for data to be read from the file. OUTPUT for data to be written to the file.
Input Field Name	`HPS_BLOB_FILE_NAME character(8)` Location of a field containing the file name. Use the LOC function to calculate a location. (Refer to the *Rules Language Reference Guide*.) Because the LOC function takes a view as its argument, the field must be the first field in a view. If the file being opened was allocated with HPS_BLOB_GENNAME_FILE, this location is the same as specified in HPS_BLOB_FILE_NAME. If the file being opened was created automatically when a BLOB file was transferred to the mainframe from another system, this is the location of an IMAGE or TEXT field in the input view of the mainframe rule receiving the BLOB file (see Transferring BLOB Files from Workstation to Mainframe)
Output View Name	`HPS_BLOB_OPEN_FILE_O`
Output Field Name	`COMPLETE small integer or integer(31)` 0 = Successful, 1 = Unsuccessful
	`HPS_BLOB_FILE_HANDLE small integer or integer(31)` File handle of the opened file. The handle is input to the following components: • HPS_BLOB_CLOSE_FILE • HPS_BLOB_DELETE_FILE • HPS_BLOB_READ_FILE • HPS_BLOB_WRITE_FILE

HPS_BLOB_READ_FILE

This section presents the purpose, the syntax, and the usage of the HPS_BLOB_READ_FILE component.

Purpose

Use this component to read data sequentially from a BLOB file. The file can be created on the mainframe with the HPS_BLOB_GENNAME_FILE component, or it can be created automatically when the file was transferred to the mainframe from another system.

Syntax

Here is the syntax of HPS_BLOB_READ_FILE component.

HPS_BLOB_READ_FILE component syntax

Name	**`HPS_BLOB_READ_FILE`**
System Identifier	`HPDREAD`

Input View Name	`HPS_BLOB_READ_FILE_I`
Input Field Name	`HPS_BLOB_FILE_HANDLE small integer or integer(31)` File handle returned by the HPS_BLOB_OPEN_FILE component in the HPS_BLOB_FILE_HANDLE field of its output view.
Input Field Name	`HPS_BLOB_LENGTH small integer or integer(31)` Length of buffer to receive data from the file. You can use the SIZEOF function to specify a length.
Input Field Name	`HPS_BLOB_BUFFER character(8)` Location of a buffer that is to receive data from the file. Use the LOC function to determine the location. (Refer to the *Rules Language Reference Guide*.) Because the LOC function takes a view as its argument, the field must be the first field in a view.
Output View Name	`HPS_BLOB_READ_FILE_O`
Output Field Name	`COMPLETE small integer or integer(31)` 0 = Successful, 1 = Unsuccessful, 4 = EOF (end of file)
Output Field Name	`HPS_BLOB_LENGTH small integer or integer(31)` Length of data read in from the file.

HPS_BLOB_WRITE_FILE

This section presents the purpose, the syntax, and the usage of the HPS_BLOB_WRITE_FILE component.

Purpose

Use this component to write data sequentially to a BLOB file. This component always appends data to the end of the file. The file can be created on the mainframe with the HPS_BLOB_GENNAME_FILE component, or it can be created automatically when the file was transferred to the mainframe from another system.

Syntax

Here is the syntax of HPS_BLOB_WRITE_FILE component.

HPS_BLOB_WRITE_FILE component syntax

Name	**`HPS_BLOB_WRITE_FILE`**
System Identifier	`HPDWRIT`
Input View Name	`HPS_BLOB_WRITE_FILE_I`
Input Field Name	`HPS_BLOB_FILE_HANDLE small integer or integer(31)` File handle returned by the HPS_BLOB_OPEN_FILE component in the HPS_BLOB_FILE_HANDLE field of its output view.
Input Field Name	`HPS_BLOB_LENGTH small integer or integer(31)` Length of buffer to be written to the file. You can use the SIZEOF function to specify a length.
Input Field Name	`HPS_BLOB_BUFFER character(8)` Location of the buffer to be written to the file. Use the LOC function to determine the location. (Refer to the *Rules Language Reference Guide*.) Because the LOC function takes a view as its argument, the field must be the first field in a view.
Output View Name	`HPS_BLOB_WRITE_FILE_O`
Output Field Name	`COMPLETE small integer or integer(31)` 0 = Successful, 1 = Unsuccessful

System Components Support Matrix

Some system components were designed for a particular environment and are not universally supported. The information in Support for System Components summarizes which system components work in which environment.

When a component does not execute immediately after a USE COMPONENT statement executes, it is referred to as Deferred. **Deferred** only applies to C language system components. See Deferred Interface System Components for more information.

Support for System Components

System Component	Java Thick Client	Java Thin Client (HTML)	3270	C Language	Deferred (C only)	C#	OpenCOBOL for Unix and Java Batch	MVS	Special	Comments
Cache_Object				X						
CDYNPIC									b	
Clear_Field_Messages[a]	X		X	X	X					
Clear_Selected_Fields	X	X		X	X					
Clear_Window_Changes	X	X	X	X	X					
DDE_ADVISE				X						
DDE_EXECUTE				X						
DDE_INITIATE				X						
DDE_POKE				X						
DDE_REQUEST				X						
DDE_TERMINATE				X						
DDE_UNADVISE				X						
Get_Altered_Field	X		X	X						
Get_Business_Process_Name	X	X		X						
Get_Current_Date_Time	X	X	X	X						
Get_Elevator_Position	X	X	X	X						
Get_First_Visible_Occurrence	X	X	X	X						
Get_Full_User_Identity	X	X	X	X						
GET_IMS_PCB_ADDR									i	
Get_Last_Visible_Occurrence	X	X	X	X		X				
Get_Listbox_Window_Sizes	X	X	X	X						
Get_Menu_Mode_By_ID	X	X	X	X						
Get_Selected_Field	X	X	X	X						
Get_Text_Message	X	X	X	X						
Get_User_Workstation_ID			X	X						
HPCLOSE								X	b	
HPDTTM									b	
HPFIND									b	
HPFREE									b	
HPFREEL									b	
HPGET									b	
HPGETL									b	
HPOPEN								X	b	
HPREAD								X	b	
HPS_BLOB_CLOSE_FILE									o	
HPS_BLOB_DELETE_FILE									o	

HPS_BLOB_GENNAME_FILE									o	
HPS_BLOB_OPEN_FILE									o	
HPS_BLOB_READ_FILE									o	
HPS_BLOB_WRITE_FILE									o	
HPS_CLOSE_FILE_LOCATE_MODE							X	X	b	
HPS_Event_Post_to_Child	X	X		X						
HPS_Event_Post_to_Parent	X	X		X						
HPS_Get_Activate	X			X		X				
HPS_Get_Environment	X	X	X	X			X	X		
HPS_Get_MinMax				X		X				
HPS_Get_Window_Handle				X						
HPS_Intercept_Events				X						
HPS_Limit_Scroll_Size				X	X					
HPS_OPEN_FILE_LOCATE_MODE							X	X	b	
HPS_READ_FILE_LOCATE_MODE							X	X	b	
HPS_Set_Activate	X			X	X	X				
HPS_Set_Bitmap_File	X			X	X	X				
HPS_Set_Cursor_By_ID	X			X	X	X				
HPS_Set_Help_Topic	X			X	X	X				
HPS_Set_HTML_File		X								
HPS_Set_HTML_Fragment		X								
HPS_Set_MinMax	X			X	X	X				
HPS_Set_Selected_Field	X	X		X	X	X				
HPS_Set_Window_Attributes				X	X					
HPS_Tbl_Get_Data_State				X						
HPS_Tbl_Init_Size				X	X					
HPS_TRUNC_FILE_LOCATE_MODE							X		b	
HPS_WRITE_FILE_LOCATE_MODE							X	X	b	
HPSETVP									b	
HPSMODE							X		b	
HPSPARM							X		b	
HPWRITE								X	b	
IMS_SPA_COMPONENT									i	
Restore_Altered_Fields	X			X	X					
Set_CloseDown_Notification	X			X	X					
Set_Control_Color_By_ID[b]	X	X	X	X	X					
Set_Control_Mode_By_ID	X	X	X	X	X					
Set_Control_RGBColor_By_ID	X	X		X	X	X				
Set_Cursor_Field	X		X	X	X					

Set_Default_Push_Button	X			X	X	X				
Set_Field_Blink_By_ID			X							
Set_Field_Color	X	X	X	X	X					
Set_Field_Message	X		X	X	X					
Set_Field_Mode	X	X	X	X	X					
Set_Field_Numeric_By_ID			X							
Set_Field_Picture	X		X	X	X	X				
Set_First_Visible_Occurrence	X	X	X	X	X					
Set_Help_File_Name	X			X	X					
SET_HPSBATCH_RETURN_CODE							X	X	b	
Set_Item_Text	X	X		X	X					
Set_Last_Visible_Occurrence	X	X	X	X	X	X				
Set_Menu_Mode	X	X	X	X	X	X				
Set_Menu_Mode_By_ID	X	X	X	X	X					
Set_PopUp_Position	X		X	X	X	X				
Set_Push_Color	X	X		X	X	X				
Set_Push_Mode	X	X	X	X	X					
Set_Virtual_Listbox_Size	X	X	X	X	X					
Set_Window_Message	X	X	X	X	X	X				
Set_Window_Position	X			X	X	X				
Set_Window_Timeout				X	X					
Set_Window_Title	X	X		X	X					
Set_Window_Title_Ex	X	X		X	X	X				
Show_Help_Topic	X			X	X					
Show_Window_Message				X		X				
Sound_3270_Alarm			X							
STOP_TP_RULE_REGION									i	
Special Components										
a: Asynch										
b: Batch										
o: BLOB										
i: IMS										

a. In Java, only edit fields are supported.
b. For both SET_CONTROL_COLOR_BY_ID and SET_CONTROL_RGBCOLOR_BY_ID, for a Thin (HTML) client there is no way to set a separate color for a single MCLB column only for the whole MCLB.

System Sets

The fields of some system components are attached to sets that are included in the default repository with the components. A list of these default or system sets appears in the following section. The system identifier of each set is in parentheses after the set name. The system identifier is the reference table name for the indicated field.

Default System Sets

The system sets are:

- EMULATOR_FKEYS (SFKEYS)
- HPS_ACTIVE_STATES (SACSTAT)
- HPS_EVENT_TYPE (SEVTYPE)
- HPS_WINDOW_STATES (SWSTATE)
- MENU_ITEM_ATTRIBUTES (SMIATTR)
- MENU_ITEM_STATE (SMIST)
- OK_YES_NO (SOKYNO)
- RETURN_CODES (SRETURN)
- SEARCH_CRITERIA (SSEARCH)
- WINDOW_OBJECT_ATTRIBUTES (SATTR)
- WINDOW_OBJECT_COLORS (SCOLORS)
- WINDOW_OBJECT_MODES (SMODE)
- WINDOW_POSITIONS (SPOPPNL)

EMULATOR_FKEYS (SFKEYS)

Used By

The values in this set may be used by 3270 Window Painter.

Values

The values of EMULATOR_FKEYS

Symbol	Value
PF1	1
PF2	2
PF3	3
PF4	4
PF5	5
PF6	6
PF7	7
PF8	8
PF9	9
PF10	Zero_A
PF11	Zero_B
PF12	Zero_C
PF13	Zero_D
PF14	Zero_E
PF15	Zero_F
PF16	10
PF17	11
PF18	12
PF19	13
PF20	14
PF21	15
PF22	16

PF23	17
PF24	18
PA2	One_E
PA3	One_F
INSERT	Two_A
DELETE	Two_B
BACK_TAB	Two_C
TAB	Two_D
NEW_LINE	Two_E
ATTN	Two_F
CLEAR	20
PRINT	Twenty_One
CUR_LEFT	Twenty_Two
CUR_RGT	Twenty_Three
CUR_UP	Twenty_Four
CUR_DOWN	Twenty_Five
CUR_HOME	Twenty_Six
CUR_D_LEFT	Twenty_Seven
CUR_D_RGT	Twenty_Eight
BKSP	Twenty_Nine
SYS_REQ	Thirty
ERA_INP	Thirty_One
ERS_EOF	Thirty_Two
IDENT	Thirty_Three
TEST	Thirty_Four
ENTER	Thirty_Nine

HPS_ACTIVE_STATES (SACSTAT)

Used By

This set is attached to the field HPS_ACTIVE_STATE, used by the following system components:

- HPS_Get_Activate
- HPS_Set_Activate

Values

The values of HPS_ACTIVE_STATES

Symbol	Value
INACTIVE	0
ACTIVE	1

HPS_EVENT_TYPE (SEVTYPE)

Used By

This set is not used for Java or C#. Instead, all symbols from this set are redefined in Constants class (see *ObjectSpeak Reference Guide* for more information) and they are available in the rule (with the same names as set symbols). The values are used to populate EVENT_TYPE field of HPS_EVENT_VIEW (see *ObjectSpeak Reference Guide*, EventType() in Converse event class).

For C, set values are used to populate EVENT_TYPE field of HPS_EVENT_VIEW.

Values

The values of HPS_EVENT_TYPE

Symbol	Value
LANDP_SYS_EVENT	6
INTERFACE_EVENT	1
SYSTEM_EVENT	0
USER_EVENT	2
LANDP_EVENT	3
LANDP_REQUEST	4

HPS_WINDOW_STATES (SWSTATE)

Used By

This set is attached to the field HPS_WINDOW_STATE, used by the following system components:

- HPS_Get_MinMax
- HPS_Set_MinMax

Values

The values of HPS_WINDOW_STATES

Symbol	Value
MINIMIZED	0
MAXIMIZED	1
NORMAL	2

MENU_ITEM_ATTRIBUTES (SMIATTR)

Used By

This set contains the available attributes of a menu item. It is attached to the field HPS_MItem_Attribute, used by the following system components:

- Get_Menu_Mode_By_ID
- Set_Menu_Mode_By_ID

Values

The values of MENU_ITEM_ATTRIBUTES

Symbol	Value
ENABLE	0
CHECK	1

MENU_ITEM_STATE (SMIST)

Used By

This set contains the possible states for a menu choice. It is attached to the field HPS_MItem_State, used by the following system components:

- Get_Menu_Mode_By_ID
- Set_Menu_Mode_By_ID

Values

The values of MENU_ITEM_STATE

Symbol	Value
HPS_OFF	0
HPS_ON	1

OK_YES_NO (SOKYNO)

Used By

This set is attached to the OK_BUTTON_YN field, used by the following system components:

- Set_Window_Message
- Show_Window_Message

Values

The values of OK_YES_NO

Symbol	Value
NO	0
YES	1

⚠ This set and the field to which it is attached are included for upward compatibility from earlier versions of the product. In the current version, you do not need to supply any value in the field OK_BUTTON_YN when you use one of the above two components.

RETURN_CODES (SRETURN)

Used By

This set is attached to the RETURN_CODE field. The DDE components and the following interface system components use this set:

The set used by DDE components and following system components

Clear_Field_Messages	Set_Field_Color
Clear_Selected_Fields	Set_Field_Message
Clear_Window_Changes	Set_Field_Mode
Get_Altered_Field	Set_Field_Picture
Get_Menu_Mode_By_ID	Set_First_Visible_Occurrence
Get_Selected_Field	Set_Help_File_Name
Get_Text_Message	Set_Item_Text
HPS_Event_Post_to_Child	Set_Last_Visible_Occurrence
HPS_Event_Post_to_Parent	Set_Menu_Mode

HPS_Get_Activate	Set_Menu_Mode_By_ID
HPS_Get_MinMax	Set_PopUp_Position
HPS_Set_Activate	Set_Push_Color
HPS_Set_Help_Topic	Set_Push_Mode
HPS_Set_MinMax	Set_Virtual_Listbox_Size
HPS_Set_Selected_Field	Set_Window_Message
Restore_Altered_Fields	Set_Window_Position
Set_CloseDown_Notification	Set_Window_Timeout
Set_Control_Color_By_ID	Set_Window_Title
Set_Control_Mode_By_ID	Show_Help_Topic
Set_Cursor_Field	Show_Window_Message

Values

Values of RETURN_CODES

Symbol	Value
FAILURE	0
SUCCESS	1

SEARCH_CRITERIA (SSEARCH)

Used By

This set contains the search criteria for list boxes and fields in a panel.It is attached to the field FIELD_OCCUR, used by the following system components:

- Get_Altered_Field
- Get_Selected_Field

Values

Values of SEARCH_CRITERIA

Symbol	Value
FIRST_SELECTION	-1
NEXT_SELECTION	0

WINDOW_OBJECT_ATTRIBUTES (SATTR)

Used By

This set of attributes is used for push buttons, boxes, and fields. It is attached to the fields FIELD_ATTR and PUSH_ATTR, used by the following system components:

- Set_Control_Color_By_ID
- Set_Field_Color
- Set_Push_Color

Values

Values of WINDOW_OBJECT_ATTRIBUTES

Symbol	Value

BACKGROUND	0
TEXT	1
BORDER	2
RESETBACKGROUND	10
RESETTEXT	11
RESETBORDER	12

WINDOW_OBJECT_COLORS (SCOLORS)

Used By

This set contains the colors for push buttons and field properties. It is attached to the field ATTR_COLOR, used by the following system components:

- Set_Control_Color_By_ID
- Set_Field_Color
- Set_Push_Color

Values

Values of WINDOW_OBJECT_COLORS

Symbol	Value
WHITE	0
YELLOW	1
BLUE	2
RED	3
Not supported	4
Not supported	5
GREEN	6
MAROON (magenta)	7
BLACK	8
AQUA (cyan)	9

WINDOW_OBJECT_MODES (SMODE)

Used By

This set contains the available window object modes. It is attached to the fields FIELD_MODE and PUSH_MODE, used by the following system components:

- Set_Control_Mode_By_ID
- Set_Field_Mode
- Set_Push_Mode

Values

Values of WINDOW_OBJECT_MODES

Symbol	Value	Note
INVISIBLE	0	
VISIBLE_DISABLED	1	

VISIBLE_ENABLED	2	
VISIBLE_READONLY	3	Used differently on Windows workstation and mainframe. See explanation below.

The value of 3 is used differently on the PC workstation (for Java and C development) than on the mainframe. The symbol name may be confusing for mainframe users.

- On the mainframe, the value of 3 means Invisible, enabled (3270 Converse only).
- On a Windows workstation, for C and Java, the value of 3 means Visible, Read-only.

WINDOW_POSITIONS (SPOPPNL)

Used By

This set contains the standard panel pop-up positions for nested windows. It is attached to the field POPTYPE, used by the following system component:

- Set_PopUp_Position

Values

Values of WINDOW_POSITIONS

Symbol	Value
ABSOLUTE_COORDINAT	1
LOGICAL_TO_WINDOW	2
LOGICAL_TO_CLIENT	3