

AppBuilder
By Magic Software Enterprises

Magic Software AppBuilder

Version 3.2

ObjectSpeak Reference Guide

Corporate Headquarters:

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

© 1992-2013 AppBuilder Solutions

All rights reserved.

Printed in the United States of America.

AppBuilder is a trademark of AppBuilder Solutions. All other product and company names mentioned herein are for identification purposes only and are the property of, and may be trademarks of, their respective owners.

Portions of this product may be covered by U.S. Patent Numbers 5,295,222 and 5,495,610 and various other non-U.S. patents.

The software supplied with this document is the property of AppBuilder Solutions and is furnished under a license agreement. Neither the software nor this document may be copied or transferred by any means, electronic or mechanical, except as provided in the licensing agreement.

AppBuilder Solutions has made every effort to ensure that the information contained in this document is accurate; however, there are no representations or warranties regarding this information, including warranties of merchantability or fitness for a particular purpose. AppBuilder Solutions assumes no responsibility for errors or omissions that may occur in this document. The information in this document is subject to change without prior notice and does not represent a commitment by AppBuilder Solutions or its representatives.

1. ObjectSpeak Reference Guide	2
1.1 Introduction to ObjectSpeak	2
1.2 User-Interface Objects	6
1.2.1 Comprehensive List of Objects	6
1.2.2 Abstract Class Objects	7
1.2.3 Basic Control Objects	17
1.2.4 Dynamic-Only Control Objects	52
1.2.5 Supporting Objects	63
1.3 Java Batch Objects	77
1.4 Events	79
1.4.1 Data Validation	79
1.4.2 ObjectSpeak Events	83
1.5 User-Interface Properties	104
1.6 Java Support Matrix	114
1.7 Supported Methods for Java Classes	122
1.8 Supported Methods in CSharp	124
1.9 Sample Code	162

ObjectSpeak Reference Guide

Introduction to ObjectSpeak

AppBuilder provides a way to interact with properties of objects and methods at execution time using extensions to the Rules Language and the application runtime system collectively called ObjectSpeak. ObjectSpeak uses an object-based Rules Language syntax to manage entities that are displayed within the runtime windows as objects. Rules coding for applications is simplified using ObjectSpeak because ObjectSpeak does not require component calls or changes to the hierarchy.

Managing Objects with ObjectSpeak

ObjectSpeak functions use more compact Rules Language syntax and minimize overall impact on the size and complexity of the application hierarchy. With ObjectSpeak, Rules Language takes a modern, object-based approach to window objects, such as edit fields, push buttons, and check boxes. For information about the Rules Language, refer to the *Rules Language Reference Guide*.

While designing a window in Window Painter, you can set the properties for the objects in a window. For more information on Window Painter, refer to the *Development Tools Reference Guide*. Understanding object properties helps you to manipulate these properties at execution time using ObjectSpeak.

To gain the basic understanding of ObjectSpeak, see the following sections:

- [Prerequisites](#)
- [General Syntax](#)

After you have understood the prerequisites and reviewed the syntax, you can review detailed information about the events that are triggered by the objects and how to respond to them in the following sections:

- [User-Interface Objects](#)
- [Java Batch Objects](#)
- [Events](#)
- [User-Interface Properties](#)



Portions of ObjectSpeak functionality are also available using standard system components provided by AppBuilder to perform standard functions. For more information, refer to the *System Components Reference Guide*.

Prerequisites

Readers of this guide should be familiar with developing applications using AppBuilder and installing the AppBuilder product. If you require information on installation procedures, see *Installation Guide for Windows*. You also require basic application development experience, familiarity with the Microsoft Windows NT/2000 operating system, and a working knowledge of distributed applications.

This guide is a companion to the *Rules Language Reference Guide*. You should have that information available because it documents extensions for different languages (Java, C#) to the Rules Language for supported ObjectSpeak entities.

Associated manuals

For more information about the concepts and overall process of developing applications, refer to the documents listed in the following table.

Documentation Set

Documentation Title	Topics
<i>Developing Applications Guide</i>	Detailed information on using AppBuilder to create applications.
<i>Deploying Applications Guide</i>	Detailed information on configuring and deploying applications.
<i>Development Tools Reference Guide</i>	Detailed descriptions of the tools used in AppBuilder to create applications.
<i>Repository Administration Guide for Workgroup and Personal Repositories</i>	Overview and specific details on configuring and managing repository-based development.
<i>Rules Language Reference Guide</i>	Detailed instructions for developing applications using Rules Language.

Terminology

ObjectSpeak uses some terminology that, though it differs from terms used in Window Painter, is more in keeping with current standard terminology. The following table explains the correspondence between these terms.

AppBuilder ObjectSpeak	AppBuilder Window Painter
Enabled	not Protected
Editable	not Read-only
Label	Static Text
ShortHelp	Status Line Help
Table	Multicolumn List Box (MCLB), Spreadsheet

General Syntax

Using the general syntax of ObjectSpeak involves:

- [Understanding Dot Notation](#)
- [Accessing Properties](#)
- [Using Pre-Defined Objects versus Dynamic Objects](#)
- [Calling Methods](#)
- [Responding to Events](#)

Understanding Dot Notation

The general syntax of an ObjectSpeak expression is shown in the example:

```
myPushButton.Visible
```

The name of the object is followed by a period, which is followed by the name of the property. In this example, the name of the object, a push button, is myPushButton. The property is Visible. By looking up the property Visible in [User-Interface Properties](#), we find that it is a Boolean, having a value of either True (for on or visible) or False (for off or hidden).

Accessing Properties

ObjectSpeak allows you to manipulate the properties of user interface objects during execution. For example, you may want to disable or enable a menu item dynamically in response to the current state of the program or protect an edit field by making it non-editable.

ObjectSpeak uses a "dot" notation to indicate that a property or method of an object is being accessed. The standard syntax for accessing the properties and methods of an object is:

Use the name of the object (*QueryButton*), followed by a period, followed by the name of the property (*Text*).

Using Get and Set Methods

For each property, ObjectSpeak uses two common methods: *Get* and *Set* . *Get* retrieves the value of the object's property and assigns a variable. This can be done simply by using the dot notation:

```
map myPushButton.Enabled to saveValue
```

If the property has a boolean type, then *IS* is used instead of *Get*. *Get*, *Set*, and *IS* are all prefixes, that is, they are followed by a property name. *Set* assigns a value for that property to the object, as shown in the following sample:

```
map False to myPushButton.Visible
```

Alternatively, you can use:

```
myPushButton.setVisible(False)
```

Examples: Modifying New and Existing Properties

The following ObjectSpeak sample changes the *Text* property of the push button called *QueryButton* to *Query* :

```
map 'Query' to QueryButton.Text
```

This example assigns a new value to a property.

The following example demonstrates how to determine the current value of a property:

```
if QueryButton.Text = 'Query'  
  use rule QUERY_RULE  
endif
```

In this example, the value of the *Text* property of the *QueryButton* object is not being changed; it is simply being referenced to see if it is equal to the string *Query* .

Using Pre-Defined Objects versus Dynamic Objects

Some window objects are pre-defined and can be referred to in the Rules Language code using their system identifier (HPSID). For example, an edit field defined in the window as *myEdit* , can be used as follows, without a declaration:

```
SET myEdit.text := 'Hello World'
```

Objects can also be declared and created dynamically, and require a declaration:

```
dcl  
  myEdit object type EditField;  
enddcl  
  
map new EditField to myEdit  
set myEdit.text := 'Hello Again'
```

When declaring these dynamic objects, avoid using the pre-defined object types?such as *EditField*, *PushButton*, and *Checkbox*?as the name of the object. Code generation generates ambiguous reference errors when it encounters these object names. If you want to use these names, refer to the *Rules Language Reference Guide* for an explanation of how to use aliases to redefine object names.



For HTML, adding an object to the window dynamically is *not* supported.

Calling Methods

Methods are functions or procedures that can be called on an object. Methods tell the object to perform a specific task.

For each of the properties common to window objects, set the value of that property by using the *Set* prefix before the name of the property. For a list of common properties, refer to [User-Interface Properties](#).

Parameters in Methods

The general syntax for an object method is as follows:

- Method parameters appear in parentheses following the method name. If there are no parameters, empty parentheses are shown.
- Within the parentheses, the parameter name is given first, followed by a colon, and then the type.
- Multiple parameters are separated by commas.
- If a method has a return value, the closing parenthesis of the parameters is followed by a colon, which is followed by the return value type.
- In most cases, to dynamically create an object, use *new ObjectName* . For such objects, the method for creating the object is not shown.
- Some objects (such as *Color*) require parameters to create a new instance. For these objects, the method for creating the instance is shown. The method name is the same as the object name, and the required parameters are indicated.
- Support for NIL representing null values: you can use NIL as a parameter of an ObjectSpeak method call for an OBJECT type. NIL is

generated as null in the resulting Java code.

Examples: Show and Set Methods

Using the Show Method

In this example, the Java client contains an object named *MessageBox* that displays a message. The *MessageBox* object has a method named *Show* that causes the message box to display.

```
dcl
  MyMessageBox object type MessageBox;
enddcl

map new MessageBox to MyMessageBox
map 'Invalid Data' to MyMessageBox.Title
map 'The amount in the Interest field is too large'
  to MyMessageBox.Message
MyMessageBox.Show
```

This example:

1. Creates a *MessageBox* object and maps it to the local variable *MyMessageBox*.
2. Assigns the title and message strings to the *Title* and *Message* properties.
3. Calls the *Show* method to display the message box.

Note

In an actual AppBuilder application, this computer code is located in the body of a procedure.

Using the Set Method

A Set method is used to assign values to properties. For example, consider the following code samples that accomplish the same result:

This line assigns (maps) a value directly to the *Text* property of the push button.

```
map 'Query' to QueryButton.Text
```

This line calls the *setText* set method of the push button, passing the new text as a parameter. In this example, *setText()* is the set method for the *Text* property.

```
QueryButton.SetText('Query')
```

Set methods take exactly one parameter, whose type is the same as that of the property. By convention, the name of a property set method is simply the property name prefixed by *Set*. For example, a string property named *Text* has a set method named *setText*, which takes a string parameter. Likewise, a Boolean property named *AutoSelect* has a set method named *setAutoSelect()* which takes one Boolean parameter, as the following code illustrates:

```
NameField.SetAutoSelect(True)
```

Although both methods (set property and map statement) can be used to change the value of a property, the set method provides a slightly more compact notation.

Responding to Events

In AppBuilder, rules that display windows are completely event-driven. This means by clicking or selecting an item, events are initialized. These events are sent to the rule, where they are optionally handled by special procedures called *event procedures*. Event procedures are defined within the rule itself using a special syntax and contain the logic that is needed to respond to the event. Events and event procedures play a central role in event-driven programming. In fact, writing event-driven programs is largely a process of determining which events you need to

respond to and writing event procedures that provide the appropriate responses. The following are some of the actions that generate events:

- Opening the window
- Closing the window
- Clicking a push button, radio button, or check box
- Selecting a menu item
- Shifting focus to a user interface object, such as an edit field
- Shifting focus away from a user interface object
- Double-clicking on an object
- Entering erroneous data into an error field
- Trying to close the window when one or more fields contain erroneous data

It is not necessary to provide an event handler for every event that is generated. If the program is required to respond to the event, an event procedure must be defined and the code that responds to the event must be placed within the procedure.

Example: Closing a Window Using an Event Procedure

If a window contains a push button named *CloseButton* that is used to close the window when clicked, the required event procedure is:

```
proc for Click object CloseButton
( e object type ClickEvent )
    MAIN_WINDOW.Terminate
endproc
```

This event procedure responds to the *Click* event by calling the *Terminate* method on the window, causing the window to close.


User-Interface Objects

User-interface objects are used to build windows for client applications on different platforms (like Java, .NET, etc.). This section discusses all of the available objects and the properties, methods, and events for ObjectSpeak objects. For each object described in this section, an overview is provided followed by lists of the properties, methods, and events for that object. Detailed information about the properties that are common to many of these objects is summarized in [User-Interface Properties](#).

The objects are organized into these categories:

- [Abstract Class Objects](#)
- [Basic Control Objects](#)
- [Dynamic-Only Control Objects](#)
- [Supporting Objects](#)

The first category includes the high-level user interface objects, including Rule and Window. The second category includes the basic building blocks of any window user interface, from check box to edit field. The third category includes the functional objects that can only be generated dynamically, such as the pop-up menu. The fourth category consists of support objects, such as colors and fonts, that support the other user interface building blocks.

 For information on how to create these objects using the Window Painter, refer to the Window Painter tool topic in the *Development Tools Reference Guide*.

Comprehensive List of Objects

The following table lists the user interface objects available in ObjectSpeak.

ObjectSpeak objects

Accelerator	Ellipse	Label	PasswordField	SetItem	Window
-----------------------------	-------------------------	-----------------------	-------------------------------	-------------------------	------------------------

CheckBox	FileEditor	ListBox	Point	System	
Color	Font	Locale	PopupMenu	Table	
Column	Format	Menu	PushButton	TabControl	
ComboBox	Formats (Derived)	MenuBar	RadioButton	TabPage	
Constants	GlobalEvent	MenuItem	Rectangle	Timer	
Dimension	GroupBox	MessageBox	Rule	TreeView	
EditField	GuiObject	MultiLineEdit	Set	TreeNode	

These objects are also categorized as:

- [Abstract Class Objects](#)
- [Basic Control Objects](#)
- [Dynamic-Only Control Objects](#)
- [Supporting Objects](#)

Abstract Class Objects

The following objects are the Abstract Class (high-level) user interface objects available in ObjectSpeak:

- [Format](#)
- [GuiObject](#)
- [Rule](#)
- [System](#)
- [Window](#)

Format

The *Format* object specifies the information needed to format text for display in various types of fields. A *Format* object is used for edit fields, combo boxes, list boxes, and table columns, where it is specified.

When a GUI object, such as an edit field, is created during the design phase in Window Painter, it automatically creates the *Format* object and generates the code to call *setFormat()* on the GUI object. The *Format* object used by a user interface object can be obtained by viewing its *Format* property.

DisplayMask specifies the mask that is used when the field does not have focus. Focus indicates that the object is active. For backwards compatibility, the *DisplayPicture* property is also included; it is equivalent to *DisplayMask*. *EditMask* specifies the mask that is used to format the text when the field has focus.

For detailed information on display and edit masks, refer to the *Developing Applications Guide*.

Properties and Methods

The following table lists the properties and methods for this object.

Format object properties and methods

Property:Type (Get Method)	Set Method
DisplayMask:String	setDisplayMask(String)
DisplayPicture:String	setDisplayPicture(String)
EditMask:String	setEditMask(String)
Type:Integer	
Additional Get Method	Additional Set Method
getDisplayString(DataObject):String	
getEditString(DataObject):String	

Example: Modifying the Display Mask

The following code sample shows how to use ObjectSpeak in the rules code to modify the display mask used by an edit field named *BirthDateField* :

```
map 'dd/mm/yyyy' to BirthDateField.Format.DisplayMask
```

The following example copies the format used by one field to another field:

```
map BirthDateField.Format to HireDateField.Format
```

GuiObject

A *GuiObject* object is a generic type used to display any interface object. Many events provide a reference to the object that triggered the event. This reference is specified in the [Source:GuiObject](#) property of the event, as illustrated in [Example: GuiObject](#). The type of Source is *GuiObject*.

Being a generic type, *GuiObject* provides properties and methods common to many interface objects but does not contain all the properties and methods available for the objects. Some of the properties and methods defined in *GuiObject* are not implemented on the actual object represented by *GuiObject*. For example, when *GuiObject* represents a menu item, calling the [Size:Dimension](#) property has no effect because you cannot specify the size of a menu item.

GuiObject cannot be used to access object properties other than those listed in [GuiObject object properties and methods](#). Thus, while a *GuiObject* may refer to an object that is a push button, you cannot access properties of the push button other than those listed.

Because *GuiObject* is a generic type that represents any GUI object, it can be used to disable an edit field or any other user interface object when it is clicked.

Properties and Methods

The following table lists the properties and methods for *GuiObject* :

GuiObject object properties and methods

Property:Type (Get Method)	Set Method
Background:Color	setBackground(Color)
Enabled:Boolean	setEnabled(Boolean)
Focus:Boolean	setFocus()
Font:Font	setFont(Font)
Foreground:Color	setForeground(Color)
HpsID:String	setHpsID(String)
Location:Point	setLocation(Point)
ShortHelp:String	setShortHelp(String)
Size:Dimension	setSize(Dimension)
Type:Integer	(Read only)
Visible:Boolean	setVisible(Boolean)

Additional Action Methods

GuiObject supports methods related to focus. The *hasFocus()* method is used to determine if the object currently has focus. (It is recommended that you use the Focus property instead of hasFocus() method). The *setFocus()* method is used to set focus to the object.

Thin Client Support

The following properties are *not* supported in thin client applications:

- [Font:Font](#)
- [Location:Point](#)
- [Size:Dimension](#)
- [Focus:Boolean](#)
- [ShortHelp:String](#)

Example: GuiObject

In the following example, an application has a Query button and an edit field in the window with hpsid, MyEdit must be disabled temporarily when it is pressed. Use the following event procedure:

```
proc QueryButtonClick for Click object QueryButton
(e object type ClickEvent)
    // find the EditField MyEdit, and disable it(the findGuiObject
// function returns a GuiObject)
thisRule.findGuiObject("MyEdit").setEnabled(false)
endproc
```

Rule

The *Rule* object plays a central role in the AppBuilder client (Java, .NET, etc.) because it provides an object interface to AppBuilder Rules Language rules. The *Rule* object has no properties, but it does define a number of methods to initiate actions, obtain information, and implement events.

This section includes:

- [Rule Methods](#)
- [postTo Methods](#)
- [Handling Cookies for Thin Client Only](#)
- [Events](#)

Rule Methods

The following table lists the methods for the *Rule* object.

Rule object properties and methods

Property:Type (Get Method)
InputView:View
OutputView:View
LongName:String
ShortName:String
ImplName:String
For Thick and Thin Clients only:
Instance:String
CallingRule:Rule
ActiveWindow:Window
Window:Window

Rule object additional methods

Additional methods
queryUserAuthentication():Boolean
setUserAuthentication(userID:String, password:String)
terminate()
trace(message:String<.view field>)
For Thick and Thin Clients only
findGuiObject(SystemID:String):GuiObject
findGuiObject(SystemID:String,type:Integer):GuiObject

postToChild(InstanceName:String, EventName:String):Boolean
postToChild(InstanceName:String, EventName:String, ViewName:View):Boolean
postToChild(InstanceName:String, EventName:String, ViewName:View, Parameter:String):Boolean
postToParent(EventName:String):Boolean
postToParent(EventName:String, ViewName:View):Boolean
postToParent(EventName:String, ViewName:View, Parameter:String):Boolean
postTo(aRule:rule, EventName:String)
postTo(aRule:rule, EventName:String, aView:View)
postTo(aRule:rule, EventName:String, aView:View, Parameter:String)
For Thick Clients Only:
setHelpFile(HelpFileName:String):Boolean
showHelpTopic(HelpID:String):Boolean
For Thin Client Only:
addCookie(CookieName:String, CookieValue:String, AgeSeconds:Integer)
getCookie(CookieName:String):String
getCookie(CookieView:View)
setCookie(CookieView:View, AgeSeconds:Integer)

InputView:View and OutputView:View

The *InputView* and *OutputView* properties return references to the input and output views. These properties are in the form of View objects.

For example:

```

dcl
    L_INPUT_V like THIS_RULE_INPUT_VIEW;
enddcl
map thisRULE.InputView to L_INPUT_V

```

LongName:String

The *LongName* property gets the long name of the rule.

ShortName:String

The *ShortName* property gets the short name of the rule.

ImpName:String

The *ImpName* property gets the implementation name of the rule.

ActiveWindow:Window

The *ActiveWindow* property returns a reference to the first window found attached to the current rule or its parents when traversing back up the rule hierarchy. When current rule is within a detached sub-process, the traversal terminates at the root of the detached sub-process, otherwise it terminates at the application's root rule. Any such window found is known as the active or target window. If no window is found, null is returned. The reference to the window is returned in the form of a Window object. If the rule displays a window, then *Window* and *ActiveWindow* properties return the same value.

```

dcl
    l_activeWindow object type Window;
enddcl
set
    l_activeWindow := thisRule.ActiveWindow
    if not isClear(l_activeWindow)
        trace(" Active window : ", l_activeWindow.Text)
    else
        trace(" No active window")
    endif

```

Window:Window

The *Window* property returns a reference to the window, if any, displayed by the rule. The reference is returned in the form of a *Window* object. If the rule does not display a window, this method returns a null reference. In order to test for this returned null reference, use the *isClear* function.

CallingRule:Rule

The *CallingRule* property returns the current rule's parent rule. Using this method allows to navigate back through the calling tree to help with many things; tracing the calling context is a good example to aid problem determination. It is not implemented in C#.

queryUserAuthentication():Boolean

The *queryAuthentication():Boolean* method forces user credentials to be retrieved by whatever mechanism specified by the AUTH_TYPE setting in the appbuilder.ini file. Any previously retrieved credentials are discarded. The method returns true if new credentials were successfully retrieved.

setUserAuthentication(userID:String, password:String)

The *setUserAuthentication()* method enables you to set the user credentials to authenticate the remote server rules. The same method is invoked if QUERY_AUTHENTICATION_ON_STARTUP is enabled through the setting in the APPBUILDER.INI file. The AppBuilder communication exits can override this information when a remote rule is invoked.

terminate()

The *terminate()* method terminates the rule.

trace(message:String<,view|field>)

The *trace(message:String)* method traces the message. This method accepts either a View or a Field as an optional second parameter. When the second parameter is specified, the name and the value of the specified object is appended to the trace message. If a View is specified as the second parameter, the name and the value of all fields are added as separate lines in the trace output.

findGuiObject(SystemID:String):GuiObject

The *findGuiObject()* method searches for an object with the given system identifier (HPSID) on the active window. The active window is the non-detached window most recently opened by a rule or its parents. The reference to the object is returned in the form of a *GuiObject*, so it can be directly manipulated using the methods defined in *GuiObject*. If there is no active window, this method returns a null reference. In order to test for this returned "null reference, use the *isClear* function.

For example:

```

dcl
    aEditField object type EditField;
enddcl
map thisrule.findGuiObject('OBJECT_HPSID') to aEditField
if isClear(aEditField)
    *> error scenario <*>
endif

```

findGuiObject(SystemID:String,type:Integer):GuiObject

Returns *GuiObject*, if the object with SystemID matches the type specified or returns null. Types constants are defined in the Constants class as follows.

Constants.WINDOW	Constants.LISTBOX
------------------	-------------------

Constants.BITMAP	Constants.MENU
Constants.CHECKBOX	Constants.MENUITEM
Constants.COLUMN	Constants.MULTILINEEDIT
Constants.COMBOBOX	Constants.PASSWORDFIELD
Constants.EDITFIELD	Constants.POPUPMENU
Constants.ELLIPSE	Constants.PUSHBUTTON
Constants.FILEEDITOR	Constants.RADIOBUTTON
Constants.GROUPBOX	Constants.RECTANGLE
Constants.LABEL	Constants.TABLE

postTo Methods

The *postTo* methods enable you to send a view to a given window.

postToChild(InstanceName:String, EventName:String):Boolean

This *postToChild()* method posts the specified event to the child with the specified instance name.

postToChild(InstanceName:String, EventName:String, ViewName:View):Boolean

This *postToChild()* method posts the specified event and the view to the child with the specified instance name.

postToChild(InstanceName:String, EventName:String, ViewName:View, Parameter:String):Boolean

This *postToChild()* method posts the specified event, the view, and the parameter to the child with the specified instance name.

postToParent(EventName:String):Boolean

This *postToParent()* method sends the specified event to the parent window.

postToParent(EventName:String, ViewName:View):Boolean

This *postToParent()* method sends the specified event and the view to the parent window.

postToParent(EventName:String, ViewName:View, Parameter:String):Boolean

This *postToParent()* method sends the specified event, the view, and the parameter to the parent window.

postTo(aRule:rule, EventName:String)

This *postTo()* method sends to itself the specified event. The first parameter of a Rule must be this rule or the long name of the rule posting event.

postTo(aRule:rule, EventName:String, aView:View)

This *postTo()* method sends to itself the specified event and the view. The first parameter of a Rule must be this rule or the long name of the rule posting event.

postTo(aRule:rule, EventName:String, aView:View, Parameter:String)

This *postTo()* method sends to itself the specified event, the view and the parameter string. The first parameter of a Rule must be this rule or the long name of the rule posting event.

setHelpFile(HelpFileName:String):Boolean

The *setHelpFile()* method specifies the name of help file that contains the help information for this rule's window. This help file also is used by windows displayed by all descendent rules of this rule, unless those rules specify their own help files.



setHelpFile() is *not* supported in thin client applications.

showHelpTopic(HelpID:String):Boolean

The `showHelpTopic()` method displays the help topic specified by the help ID from the window's help file. As noted above, if this window has not specified a help file, it uses the help file of its parent rule. For windows, the help ID must be the long name of the window. For user interface objects placed on the window, the ID must be the window long name, followed by a dot, followed by the system identifier (HPSID) of the object.



`showHelpTopic()` is *not* supported in thin client applications, neither in .NET applications.

Handling Cookies for Thin Client Only

The Rule object supports these additional methods for thin client only:

- [addCookie\(CookieName:String, CookieValue:String, AgeSeconds:Integer\)](#)
- [getCookie\(CookieName:String\):String](#)
- [setCookie\(CookieView:View, AgeSeconds:Integer\)](#)
- [getCookie\(CookieView:View\)](#)

addCookie(CookieName:String, CookieValue:String, AgeSeconds:Integer)

The `addCookie(CookieName:String, CookieValue:String, AgeSeconds:Integer)` method allows any valid cookie name as a name (as opposed to the old `setCookie`, which allows only the system FIELD names as names of the cookie). With this method, you can have a cookie like "Company.Dept.Developer" as a name with dots, etc., as part of the name. The value is any valid string value and the age is the expiry time of the cookie in seconds. If the expiry time is bigger than 0 the cookie is available for all browser sessions until it expires. If the expiry time is 0 the cookie is local to the browser session in which it was created and does not expire until that session is closed.

getCookie(CookieName:String):String

The `getCookie(CookieName:String)` method returns the data associated with the given cookie name.

setCookie(CookieView:View, AgeSeconds:Integer)

The `setCookie(CookieView:View, AgeSeconds:Integer)` can be used to create client-side HTTP cookie with the names of fields in the view and values of the fields making up the name-value pair elements of the cookie. The age sets up the expiry time in seconds. As the HTTP cookie does not support hierarchical data structures, this method uses only FIELDS directly under the given View; all nested views are ignored. If the expiry time is bigger than 0 the cookie is available for all browser sessions until it expires. If the expiry time is 0 the cookie is local to the browser session in which it was created and does not expire until that session is closed.

getCookie(CookieView:View)

The `getCookie(CookieView:View)` retrieves cookie data sent to the web client and is saved on the web client environment for the specified time. Subsequent requests from this client environment receive the cookie data and the AppBuilder rule invokes `getCookie(aView:View)` to retrieve the data. The cookie data with names matching the FIELD names of this view are read into the view. For example:

```
AB_WEBCLIENT_RULE.setCookie(AB_WEBCLIENT_VIEW, 3600)
```

where `AB_WEBCLIENT_RULE` is the name of the rule, `AB_WEBCLIENT_VIEW` is the view containing the set of fields to be used as cookie names and values, and `3600` is the expiry time of the cookies in seconds.

```
AB_WEBCLIENT_RULE.getCookie(AB_WEBCLIENT_VIEW)
```

where `AB_WEBCLIENT_RULE` is the name of the rule and `AB_WEBCLIENT_VIEW` is the view containing the set of fields to be filled with data if the field names match the cookie names.

Events

The following events can be triggered using the *Rule* object:

- [Activate](#)
- [ChildRuleEnd](#)
- [CommError](#)
- [Converse](#)
- [Initialize \(for Rule\)](#)
- [ParentRuleEnd](#)
- [Post](#)
- [RuleEnd](#)
- [SQLException](#)
- [Terminate \(for Rule\)](#)

Activate, *ChildRuleEnd*, *ParentRuleEnd*, *RuleEnd*, and *SQLException* events exist but currently are never triggered.

The *ChildRuleEnd* event is triggered when a child rule of the current rule has terminated. The *RuleEnd* event is triggered when another rule, for which the current rule has registered an event procedure, terminates.

The rule that ends must be a detached rule that was started using the command:

```
use rule < rule_name > detach < rule_object_name >
```

where *< rule_object_name >* is the name of the local variable (defined in the *dcl* section) which references the rule.

The *Converse* event on the rule is the same as the *Converse* event on the window; only one event needs to be registered, either the rule's or the window's. The *Initialize (for Rule)* and *Terminate (for Rule)* events are distinct from those for the Window object. The *Post* event is triggered when the rule receives a view that was posted by another rule using one of the two *Post* methods described in [Rule Methods](#).

System

The System object provides an interface for some system services. Two methods are used to translate long names of AppBuilder entities, such as rules or views, into corresponding Java class names or Java object names according to the AppBuilder naming conventions. It is not supported for .NET clients.

System object is not supported in C#.

Constants and Methods

Methods:

[longNameToClassName\(type:Integer, longName:String\) : String](#)

[longNameToObjectName\(type:Integer, longName:String\) : String](#)

Type values:

The following table shows the various system type value

System type values	
RULE_TYPE	COMPONENT_TYPE
VIEW_TYPE	WINDOW_TYPE
VIEWARRAY_TYPE	OBJECT_TYPE
SET_TYPE	HPSID_TYPE
FIELD_TYPE	

longNameToClassName(type:Integer, longName:String) : String

The *longNameToClassName* method translates the long name of an AppBuilder entity, whose type is specified by the first parameter using one of the System object constants, into a corresponding Java class name according to AppBuilder naming conventions.

For example:

```
SET RuleClassName := System.longNameToClassName(System.RULE_TYPE, "MY_RULE")
RuleCaller.ExecuteRule(RuleClassName)
```

The *type* parameter can have only one of the following values: RULE_TYPE, VIEW_TYPE, VIEWARRAY_TYPE, SET_TYPE and COMPONENT_TYPE; other entities do not generate a class.

longNameToObjectName(type:Integer, longName:String) : String

The *longNameToObjectName* method translates the long name of an AppBuilder entity, whose type is specified by the first parameter using one of the System object constants, into a corresponding Java object name according to the AppBuilder naming conventions.

For example:

```
SET RuleClassName := System.longNameToClassName(System.RULE_TYPE, "MY_RULE")
SET ViewObjectName := System.longNameToObjectName(System.VIEW_TYPE, "MY_VIEW")
```

RuleClassName ++ *ViewObjectName* forms a reference to a field of generated rule class, which corresponds to the instance of view MY_VIEW

owned by MY_RULE.

The type parameter can be anything except COMPONENT_TYPE. This is because components' classes are never instantiated, and there is no corresponding property in the rule class. Use HPSID_TYPE for windows' objects that have HPSID, but OBJECT_TYPE for other objects and aliases.

Window

The *Window* object plays a central role in the client (Java, .NET, etc.) because it provides an object interface to windows. The *Window* object has numerous properties, defines several methods to initiate actions or obtain information, and implements several events.



The window panel is set to the resolution of the window at the time the panel is designed. This could differ from the resolution at execution time, thereby causing unexpected display properties.

This section includes the following topics:

- [Properties and Methods](#)
- [Events](#)


Properties and Methods

The following table describes the properties and methods for the *Window* object.

Window object properties and methods

Property:Type (Get Method)	Set Method
Altered: Boolean	setAltered(Boolean)
Background: Color	setBackground(Color)
DefaultButton: PushButton	setDefaultButton(PushButton)
Foreground: Color	setForeground(Color)
FocusedGuiObject: GuiObject	(read only)
HpsID: String	(read only)
Location: Point	setLocation(Point)
MenuBar: MenuBar	setMenuBar(aMenuBar:MenuBar)
PopupMenu: PopupMenu	setPopupMenu(PopupMenu)
Resizable: Boolean	setResizable(Boolean)
Size: Dimension	setSize(Dimension)
Text: String1	setText(String)
Type: Integer	(Read only)
Visible: Boolean	setVisible(Boolean)
Additional Get Method	Additional Set Method
clearSelection()	
getHelpTopic():String	
	addChild (Object)*
	clearAltered()
	clearWindowChanges()
	printFrame()
	ShowMessageBox(messageType:Integer, message:String):Integer
	ShowMessageBox(message:String, title:String, buttonType:Integer, messageType:Integer):Integer*

	terminate()
	updateDisplay()*

 The Text property can be used to get/set the Title of the window.

clearSelection()

This method clears selected items of all list boxes and Tables (MCLB) in the window.

DefaultButton:PushButton


This property specifies the default pushbutton for the window.

getHelpTopic():String

This method returns HelpTopic name, if a Helpset is specified.

Resizable:Boolean

The *Resizable* property specifies whether the window can be resized by the user during execution.


 This property is *not* supported for thin (HTML) client.

In Java, if a window is resizable, the minimize and maximize buttons appear at the right of the title bar. If the window is not resizable, these buttons are not available and the window cannot be minimized or maximized.


In C#, if a window is resizable then a resizable border will be drawn, otherwise a three-dimensional border will be drawn.

Additional Action Methods

- The *addChild()* method adds a programmatically-created object to the window (either GUI Objects or Java Beans).

 Not supported for thin client (HTML). In thin client, GUI objects cannot be dynamically created and added using *addChild*. Refer to the System Components Reference Guide for information about using `HPS_SET_HTML_FILE` and `HPS_SET_HTML_FRAGMENT`.

- The *terminate()* method terminates the current window, allowing the owning rule to return.

 Not supported for thin client (HTML).

- The *updateDisplay()* method causes the visible window to immediately repaint itself. This method is useful for replacing the "converse < window > nowait" statement from the earlier converse-driven versions of the product. If the window has not yet been displayed, call the *setVisible(True)* method instead to display the window. The *updateDisplay()* method is called implicitly when the [Visible:Boolean](#) property is set to *True*.
- The *showMessageBox()* method can be in either:

```
ShowMessageBox (message: String, title: String, buttonType: Integer, messageType: Integer)
```

or

```
ShowMessageBox (messageType: Integer, message: String)
```

where *message* is the message to show, *title* is the title for the message box, *buttonType* is one of the following:

- `Constants.DEFAULT_BUTTONS`
- `Constants.OK_BUTTON`
- `Constants.OK_CANCEL`
- `Constants.YES_NO`

- *Constants.YES_NO_CANCEL* and *messageType* is one of the following:
- *Constants.ERROR*
- *Constants.INFORMATION*
- *Constants.WARNING*
- *Constants.QUESTION*
- *Constants.PLAIN*

In C#, constant values will be replaced with appropriate enum values, *ButtonType* will be replaced with *MessageBoxButtons*, and *messageType* will be replaced with *MessageBoxIcon*.



Not supported for thin client (HTML).

Events

The following events can be triggered by the *Window* object:

- [Close](#)
- [Converse](#)
- [Initialize \(for Window\)](#)
- [Terminate \(for Window\)](#)
- [WindowError](#)
- [WindowValidation](#)

The [Close](#) event is triggered when the user attempts to close the window using the system exit (the **X** in the upper right corner of the window) or uses the system hot key (usually **Alt+F4**). The system does nothing by default. If the system is to shut down, you must explicitly terminate the window. Refer to the catalog of events (the Close Window with System Menu event, in particular) in the *Developing Applications Guide*. This event has no methods or properties.

The [Initialize \(for Window\)](#) event is triggered after the window is created but before it is shown. It is an ideal place to initialize the program's data, as well as to make any needed adjustments to the visual objects on the window. The [Terminate \(for Window\)](#) event is called while the window is closing. These events are distinct from those for the Rule object.

The [WindowError](#) and [WindowValidation](#) events are called when the user attempts to close the window with a push button or menu item for which validation is enabled. Window validation is used to prevent the window from closing when editable fields contain invalid information, when mandatory fields are empty, and for any other condition which the application logic concludes is unacceptable.

Thin Client Support

In thin client development, the following events are *not* supported:

- [Close](#)
- [WindowError](#)
- [WindowValidation](#)

In thin client applications, [WindowError](#) and [WindowValidation](#) can be done through JavaScript using the *extension.js*, which can be customized to include any [WindowError/Validation](#) handling procedures.

For thin client applications only, a [MessageBox](#) Event is issued when a [showMessageBox](#) is called on the window with the *MessageType* set to *QUESTION*. All other types than *QUESTION* will show a messagebox with a warning icon.

Basic Control Objects

The following table lists the basic control objects for a user interface that can be defined in the Construction Workbench Window Painter as static objects or during execution using *ObjectSpeak* in rules source code as dynamic objects.

Control objects

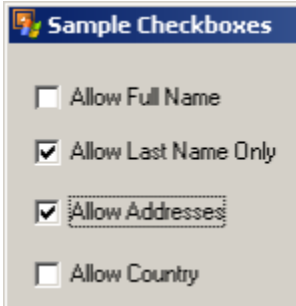
CheckBox	FileEditor	MenuBar	RadioButton
Column	GroupBox	MenuItem	Rectangle
ComboBox	Label	MultiLineEdit	TabControl
EditField	ListBox	PasswordField	Table
Ellipse	Menu	PushButton	TabPage

CheckBox

The *CheckBox* object displays the standard check box object allowing end users to specify a yes/no or on/off condition for a setting. This object

can be created dynamically during execution and added to the window in Rules Language code as described in the following section. For a code sample, refer to [Sample Code](#). The following figure shows a sample dialog with check boxes.

Sample dialog with check boxes



Constructor and Parameters

The following code is a sample declaration and construction:


```

dcl
aCheckBox object type CheckBox;
enddcl

map NEW CheckBox() to aCheckBox
    
```

There are no parameters for this object.

Properties and Methods

 Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists own properties and methods for the *CheckBox* object:

Checkbox object properties and methods

Property:Type (Get Method)	Set Method
Altered: Boolean	setAltered(Boolean)
AutoSelect: Boolean	
DataLink: DataObject	setDataLink(DataObject)
ImmediateReturn: Boolean	setImmediateReturn(Boolean)
Mnemonic: Char	setMnemonic(Char)
MnemonicKeycode: Integer	setMnemonicKeycode(Integer)
PopupMenu: PopupMenu	setPopupMenu(PopupMenu)
Selected: Boolean	setSelected(Boolean)
TabStop: Boolean	setTabStop(Boolean)
Text: String	setText(String)

DataLink: DataObject

The check box can be linked to a character field of length one (1) that contains an X when the check box is selected, or to a Boolean field that contains TRUE if the check box is selected.

The DataObject for a check box is Boolean or string.

Set the property as shown in the following example:

```
aCheckBox.setDataLink( field_of_View_of_View )
```

Additional Action Methods

The *CheckBox* object has methods related to focus. The *hasFocus()* method is used to determine if the object currently has focus. The *setFocus()* method is used to request that focus be set to the object.

Events

The following events can be triggered by the *CheckBox* object:

- [Click](#)
- [DoubleClick](#)
- [FocusGained](#)
- [FocusLost](#)

Click

The *CheckBox* object triggers the [Click](#) event when any of the following occur:

- The user presses the primary mouse button when the mouse is on the object.
- The spacebar is pressed when the object has the focus.
- The user presses the object's mnemonic key (Alt+mnemonic key).
- The [Selected:Boolean](#) property is changed.
- A value is mapped into the field data-linked to the object.

The *CheckBox* object triggers the [FocusGained](#) event when the following occur:

- The user tabs into the object or clicks on the object.

The *CheckBox* object triggers the [FocusLost](#) event when the following occur:

- The user tabs out of the object or clicks on another object in the window or on the window itself.

Column

The *Column* object plays a fundamental role in implementing tables in the client. In general, a [Table](#) object owns one or more *Column* objects. The *Table* object has a number of properties that affect the table as a whole, while the *Column* object contains a number of properties that determine the appearance and behavior of individual columns. The properties of a column affect all cells within that column.



The *Column* object is not a stand-alone object you *cannot* place a *Column* object into a window without a [Table](#) object.

You can access properties or methods on the table or on individual columns of the table. As with all other visual objects, the names of tables and columns are the same as their system identifier (HPSID).

The table is data-linked to an occurring view and each column in the table (other than the optional numbering column) is associated with a field in the hierarchy beneath the occurring view. The complete data link consists of a data link from the table to the occurring view. For each column, data is linked using a specification of the path from the occurring view to the particular field in the view to which the column is linked. These data links are specified using Set methods with the following properties:

- `< columnname >.FieldPath` path from occurring view to field
- `< tablename >.ViewLink` link to the occurring view

The first is set with the [Table](#) object. The second is set with [Column](#) object.

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for the *Column* object:

Column object properties and methods

Property:Type (Get Method)	Supported Set Method
Altered: Boolean	setAltered(Boolean)
Editable: Boolean	setEditable(Boolean)
EditLimit: Integer	setEditLimit(Integer)
Error: Boolean	(read only)
Empty: Boolean	(read only)
FieldPath: String	setFieldPath(String)
Format: Format	setFormat(Format)
HeaderLineCount: Integer	(read only)
ImmediateReturn: Boolean	setImmediateReturn(Boolean)
Justification: Integer	setJustification(Integer)
Mandatory: Boolean	setMandatory(Boolean)
PopupMenu: PopupMenu	setPopupMenu(PopupMenu)
SetLink: Set	setSetLink(Set)
Width: Integer	setWidth(Integer)
Additional Get Method	Additional Set Method
getHeader(): String	
getHeader(n: Integer)	
getScaledWidth(n: Integer)	
	addHeader(String) setHeader(n: Integer, value: String) removeHeader(n: Integer)
	setFieldPath(String Array)
	setScaledWidth(n: Integer)

getHeader():String

This method returns the header as a string.

getHeader(n:Integer)

This method returns the *n* th line of header string for a multi-line header.

getScaledWidth():Integer

This method returns the width of the column with respect of current coordinate system.

setFieldPath:String Array

This is the path from an occurring view to a field when linked to an occurring view. This specifies the path in the hierarchy from the view to a specific field.

For example, if the table is linked to the occurring view *MYTABLE_OCC* and the column is linked to the field *MYTABLE_COLUMN1* of *MYTABLE_OCC*, the field path will be the dynamic array with the value '*MYTABLE_COLUMN1*'.

```
dcl
  Coll_FieldPath object array of varchar(20);
enddcl
Coll_FieldPath.append('MYTABLE_COLUMN1')
COL1.setFieldPath(Coll_FieldPath)
```

FieldPath:String

This property uses String instead of String Array for FieldPath; the path is separated by dots(.). In the following example, TABLE_OCC is an occurring view linked to a table, TABLE_VIEW1 is a child view of TABLE_OCC, and COL_FIELD1 is a Field of TABLE_VIEW1 and is linked to the column COL1.

```
COL1.setFieldPathString("TABLE_VIEW1.COL_FIELD1")*
```

ViewLink:Array

This is the link for the list when linked to an occurring view. For a Column, this specifies the occurring view that contains the data to be displayed.

Width:Integer

This property the width of a table column.

The minimum width that can be set for a column in an MCLB is determined by Java, defaulting to 15 in Sun's Java 5.

setScaleWidth(n:Integer)

This method sets the width of the column with respect of current coordinate system.

Additional Action Methods

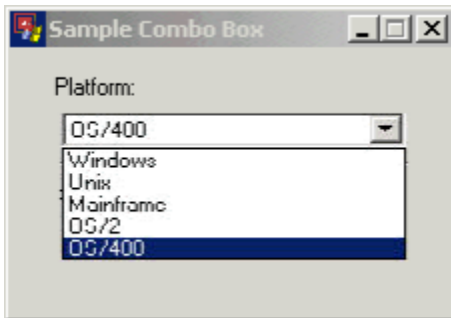
Each column can have one or more rows of header text. Each row is added by calling the *addHeader()* method, with the first call to that method specifying the first line of header text, the second call specifying the second line, and so on.

When the Column is enabled, the *setSetLink(Set)* method creates a drop-down combo box editor for the column with the values from the set. When the Column is disabled, it shows a protected edit field that displays the domain value from the set.

ComboBox

The *ComboBox* object is used to display the standard combo box object that combines an edit area and a drop-down list. The list contains choices that can be selected and that subsequently appear in the edit area.

Sample combo box



If a combo box is painted as DropDown, it is an editable combo box. If a combo box is painted as DropDownList, it is a non-editable combo box.

A Simple combo box is a combobox where the domain the list of values is permanently displayed (dropped-down), and there is no button attached to the combobox to display the list of values. Simple comboboxes are not supported in Java, neither in C#.

In a DropDown combo box, the contents of the edit area can be directly modified. In a DropDownList combo box, the contents of the edit area can be modified only by selecting items in the list. During execution, the combo box can be changed from editable to non-editable using the Editable property. To make the combo box non-editable at execution time, set the Editable property to *False*.

In an editable combo box, if text has been entered but not committed to the data link, it can be rolled back by pressing Escape (**Esc**). Pressing **Enter** while focus is on an edit area whose contents have been modified commits the changes to the data link. Moving focus away from the combo box also commits the changes.


Combo boxes can be linked to character fields, numeric fields (integer or decimal), date fields or time fields. Combo boxes can be more complex than other window objects because a combo box has two data-links: one for the edit area and one for the list. Furthermore, the list can be linked to either of two data structures: an occurring view or a set. For a combo box linked to an occurring view, the data link for the list consists of two parts: the link to the occurring view and the path in the hierarchy from the occurring view to the field itself.

Constructor and Parameters


The following is a sample declaration and construction:

```
dcl
  aComboBox object type ComboBox;
enddcl
map NEW ComboBox() to aComboBox
```

There are no parameters for this object.

 Editable objects generated in the rule source code do not have a format object defined. A format object needs to be defined and passed to the combo box if formatting is required.


Properties and Methods

 Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for the *ComboBox* object:

ComboBox object properties and methods

Property:Type (Get Method)	Set Method
Altered:Boolean	setAltered(Boolean)
AutoSelect:Boolean	setAutoSelect(Boolean)
DataLink:DataObject	setDataLink(DataObject)
DomainType:Integer	
Editable:Boolean	setEditable(Boolean)
EditLimit:Integer	setEditLimit(Integer)
Error:Boolean	(read only)
Empty:Boolean	(read only)
FieldPath:String	setFieldPath(String)
Format:Format	setFormat(Format)
ImmediateReturn:Boolean	setImmediateReturn(Boolean)
Justification:Integer	setJustification(Integer)
Mandatory:Boolean	setMandatory(Boolean)
PopupMenu:PopupMenu	setPopupMenu(PopupMenu)
SetLink:Set	setSetLink(Set)
SelectedIndex:Integer	setSelectedIndex(Integer)
TabStop:Boolean	setTabStop(Boolean)
Text:String	setText(String)
ViewLink:Array	setViewLink(Array)

 EditLimit, Empty, and Error are only valid for an editable combo box.

DataLink:DataObject

This is the link for the edit area of the combo box. This property specifies the data field in the application hierarchy to which the edit field is linked.

The DataObject for a combo box is: Date, Decimal, Integer, ShortInteger, String, or Time.

setFieldPath:String Array

This is the path from an occurring view to a field when linked to an occurring view. It specifies the path in the hierarchy from the view to a specific field.

FieldPath:String

This is the path from an occurring view to a field when linked to an occurring view. It specifies the path in the hierarchy from the view to a specific field.

SetLink:Set

This is the link for the list when linked to a set. For static combo boxes, it specifies the set from which the values displayed in the combo box are taken.

ViewLink:Array

This is the link for the list when linked to an occurring view. For dynamic combo boxes, it specifies the occurring view from which the items in the drop down list are taken.

Additional Action Methods

The *ComboBox* object has methods related to focus. The `hasFocus()` method is used to determine if the object currently has focus. The `setFocus()` method is used to request that focus be set to the object.

Events

The following events can be triggered by the *ComboBox* object:

- [Click](#)
- [DoubleClick](#)
- [FieldError](#)
- [FieldValidation](#)
- [FocusGained](#)
- [FocusLost](#)



These events are not supported for thin client combo boxes.

The *ComboBox* object triggers the [Click](#) event when any of the following occur:

- The user presses the mouse button while the mouse is on the object.
- The spacebar is pressed while the object has the focus.
- The user presses the object's mnemonic key (Alt+mnemonic key).
- The [Selected:Boolean](#) property is changed.
- A value is mapped into the field data-linked to the object.

The *ComboBox* object triggers the [FocusGained](#) event when the following occurs:

- The user tabs into the object or clicks on the object.

The *ComboBox* object triggers the [FocusLost](#) event when the following occurs:

- The user tabs out of the object or clicks on another object in the window or on the window itself.

Editable combo boxes provide for data validation using the [FieldError](#) and [FieldValidation](#) events. When an attempt is made to commit data, either by moving focus or pressing **Enter**, and the data contains errors, the [FieldError](#) event is triggered. If there are no errors, the [FieldValidation](#) event is triggered, allowing the application to verify that the data is acceptable.

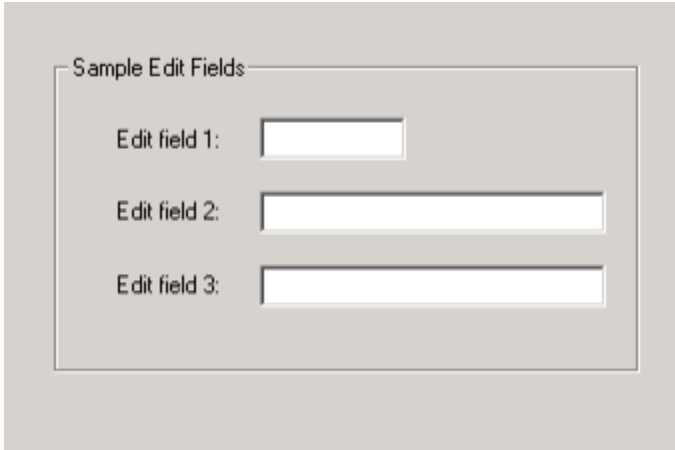
EditField

The *EditField* object displays a rectangular area into which a single line of text can be entered.

When text is entered but not committed to the data link, click **Esc to roll it back**. Pressing **Enter** when you focus on an edit field whose contents have been modified commits the changes to the data link. Moving focus away from an edit field also commits the changes.

The following figure shows a sample dialog with three edit fields. Each field has static text added in front of it.

Edit field dialog



The [PasswordField](#) object is very similar but displays asterisks (*) instead of the value. For an edit field with multiple lines, refer to the [MultiLineEdit](#) object.

Constructor and Parameters

The following is a sample declaration and construction:

```
dcl
  aEditField object type EditField;
enddcl
map NEW EditField() to aEditField
```

There are no parameters for this object.

i Editable objects generated in the rule source code do not have a format object defined. A format object needs to be defined and passed to the list box if formatting is required.

Properties and Methods

i Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object.

EditField object properties and methods

Property:Type (Get Method)	Set Method
Altered:Boolean	setAltered(Boolean)
AutoSelect:Boolean	setAutoSelect(Boolean)
AutoTab:Boolean	setAutoTab
Border:Boolean	setBorder(Boolean)
DataLink:DataObject	setDataLink(DataObject)
Editable:Boolean	setEditable(Boolean)
EditLimit:Integer	setEditLimit(Integer)
Error:Boolean	(read only)
Empty:Boolean	(read only)

Format:Format	setFormat(Format)
ImmediateReturn:Boolean	setImmediateReturn(Boolean)
Justification:Integer	setJustification(Integer)
Mandatory:Boolean	setMandatory(Boolean)
PopupMenu:PopupMenu	setPopupMenu(PopupMenu)
SetLink:Set	setSetLink(Set)
TabStop:Boolean	setTabStop(Boolean)
Text:String	setText(String)

Focus:Boolean

This method has no parameters.

AutoTab:Boolean

In an edit field, this indicates whether an automatic tab event is generated at the input of the last character of the maximum allowed limit, depending on the edit limit value or the length of the data object linked to the field. The default property is *False* which indicates the focus remains in the edit field at the input of the last allowable character and the user has to manually generate a shift of focus. This property is used only by Java runtime.

DataLink:DataObject

An edit field can be linked to a character field, a numeric field (integer or decimal), a date field, or a time field. It is not necessary for an edit field to have a data link. If it has a data link, the text in the edit field can be accessed either through the Text property or using rules code to access the data-linked field. If it does not have a data link, the only way to set or query the text in the edit field is with the Text property. The DataObject for an edit field is Date, Decimal, Integer, ShortInteger, String, or Time.

Additional Action Methods

The *setBorder()* method is used to draw the border around the edit field that makes it appear either as a three-dimensional box or a two-dimensional box depending on the 3D property, or to have no border. For example:

```

EditField.setBorder(flag:Boolean)

```

When the flag is *True*, it has a three-dimensional or two-dimensional border around the edit field. When the flag is *False*, no border is drawn around the edit field.

This object has methods related to focus. The *hasFocus()* method determines if the object currently has focus. The *setFocus()* method requests that focus be set to the object.

When the EditField is disabled, the *setSetLink(Set)* method displays the domain value from the set.

Events

The *EditField* object can trigger the following events:

- [Click](#)
- [DoubleClick](#)
- [FieldError](#)
- [FieldValidation](#)
- [FocusGained](#)
- [FocusLost](#)

This object triggers the [Click](#) event when any of the following occur:

- The user presses the primary mouse button when the mouse is on the object.

This object triggers the [FocusGained](#) event when the following event occurs:

- The user tabs into the object or clicks on the object.

This object triggers the [FocusLost](#) event when the following event occurs:

- The user tabs out of the object or clicks on another object in the window or the window itself.

EditField object provides for data validation using the [FieldError](#) and [FieldValidation](#) events. When an attempt is made to commit the data either by moving focus or pressing **Enter**, and the data contains errors, the *FieldError* event is triggered. If there are no errors, the *FieldValidation* event is triggered, allowing the application to verify that the data is acceptable.

Thin Client Support

Click, DoubleClick, FocusGained, FocusLost events are not supported for thin client *EditField*.

The *FieldError* and *FieldValidation* events are supported through client-side JavaScripts.

Ellipse

An *Ellipse* object is used to display an ellipse or circle.

Thin Client Support

This is *not* supported for thin clients (HTML).


Constructor and Parameters

The following is a sample declaration and construction:

```
dcl
  anEllipse object type Ellipse;
enddcl
map NEW Ellipse() to anEllipse
```

There are no parameters for this object.

Properties and Methods

 Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following section contains the properties and methods for this object:

Ellipse object properties and methods

The *Ellipse* class contains no properties or methods except the properties or methods of the parent class.

Events

- [Click](#)
- [DoubleClick](#)


This object triggers the [Click](#) event when any of the following occur:

- The user presses the primary mouse button when the mouse is on the object.

FileEditor

The *FileEditor* object allows end users to view, but not modify, the contents of a text file. The File editor *cannot* be made Editable.

Properties and Methods

 Inherits [GuiObject](#) and [MultiLineEdit](#) and exposes all their properties and methods (not listed below).

The following table lists the properties and methods for this object.

FileEditor object properties and methods

Property: Type (Get Method)	Set Method
-----------------------------	------------

DataLink:String	setDataLink(String)
WordWrap:Boolean	setWordWrap(Boolean)



This is *not* supported for thin client (HTML).

DataLink:String

For *FileEditor*, the data link specifies a valid file name to load into the view. The file name can be specified as a full path to the file:

```
c:\appbuilder\hps.ini
```

or just the name of the file, if the file is in the class path.

WordWrap:Boolean

This property specifies whether or not the text is wrapped when it reaches the right edge of the edit area. The default value is *TRUE*.

Events

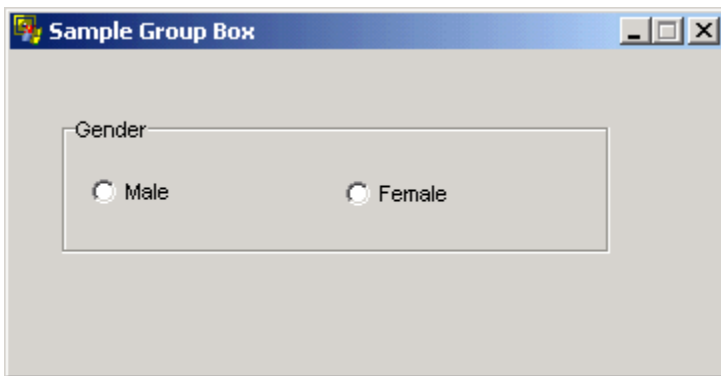
The *FileEditor* object triggers the following events:

- [Click](#)
- [DoubleClick](#)
- [FieldError](#)
- [FieldValidation](#)
- [FocusLost](#)
- [FocusGained](#)

GroupBox

A *GroupBox* object is used to display a rectangle with an optional title in the upper left, that is used to group together other objects, for example, radio buttons or check boxes.

GroupBox objects with two radio buttons



Constructor and Parameters

The following is a sample declaration and construction:

```
dcl
  aGroupBox object type GroupBox;
enddcl
map NEW GroupBox() to aGroupBox
```

There are no parameters for this object.

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object.

GroupBox properties and methods

Property:Type (Get Method)	Set Method
Text:String	setText(String)

Events

This object generates no events.

Label

The *Label* object is used to display both non-editable text and graphics in the window. Although the user cannot edit the static text, the text can be modified with rules code. A *Label* object cannot be disabled and it cannot receive focus.

Constructor and Parameters

The following is a sample declaration and construction:

```
dcl
    aLabel object type Label;
enddcl
map NEW Label() to aLabel
```

There are no parameters for this object.

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object:

Label object properties and methods

Property:Type (Get Method)	Set Method
HorizontalTextPosition:Integer	setHorizontalTextPosition(Integer)
Image:String	setImage(String)
Justification:Integer	setJustification(Integer)
Mnemonic:Char	setMnemonic(Char)
MnemonicKeycode:Integer	setMnemonicKeycode(Integer)
PopupMenu:PopupMenu	setPopupMenu(PopupMenu)
Text:String	setText(String)
VerticalJustification:Integer	setVerticalJustification(Integer)
VerticalTextPosition:Integer	
Property:Type (Get Method)	Set Method
getJustification():Integer	

getVerticalTextPosition():Integer	
	setAutoSize(Boolean)
	setVerticalTextPosition(Integer)

For a *Label*, the horizontal justification of the text within the label area is specified by the Justification property, while the vertical justification is determined by the VerticalJustification property.

VerticalJustification:Integer

For *Label* objects, the VerticalJustification property specifies vertical justification. Valid values are defined in the Constants class as:

- *TOP*
- *CENTER*
- *BOTTOM*

The default vertical justification is CENTER.

Events

The *Label* object triggers the following events:

- [Click](#)
- [DoubleClick](#)

 Not supported for thin client.

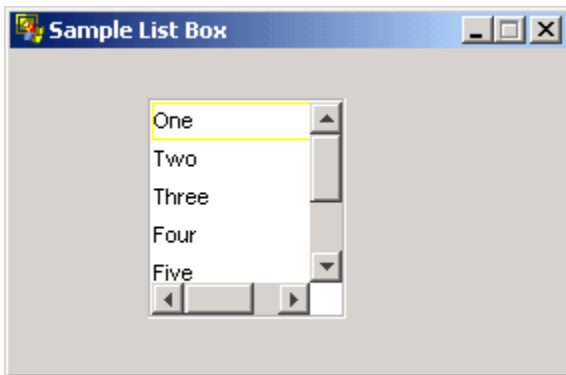
This object triggers the [Click](#) event when any of the following occur:

- The user presses the primary mouse button when the mouse is on the object.
- The user presses the object's mnemonic key (that is, Alt+mnemonic key).

ListBox

The *ListBox* object is used to display standard list box functionality. It provides a list of items displayed in a rectangular area. If the vertical size of the area is not sufficient to show all the choices, then a vertical scroll bar automatically appears. The following figure shows a sample list box.

Sample list box



List boxes can be linked to character fields, numeric fields (integer or decimal), date fields, or time fields. The data link is specified by a combination of two properties. The FieldPath property specifies the location of the specific field that contains the items relative to the view. The ViewLink property specifies occurring view that holds the items to be displayed in the list box.

Constructor and Parameters

The following is a sample declaration and construction:

```

dcl
  aListBox object type ListBox;
enddcl
map NEW ListBox() to aListBox

```

There are no parameters for this object.



Editable objects generated in the rule source code do not have a format object defined. A format object needs to be defined and passed to the list box if formatting is required.

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object:

ListBox object properties and methods

Property:Type (Get Method)	Set Method
Altered:Boolean	setAltered(Boolean)
AutoSelect:Boolean	setAutoSelect(Boolean)
Editable:Boolean	setEditable(Boolean)
Empty:Boolean	(read only)
Error:Boolean	(read only)
FieldPath:String	setFieldPath(String)
Format:Format	setFormat(Format)
ImmediateReturn:Boolean	setImmediateReturn(Boolean)
Justification:Integer	setJustification(Integer)
Mandatory:Boolean	setMandatory(Boolean)
NextSelectedIndex:Integer	(read only)
PopupMenu:PopupMenu	setPopupMenu(PopupMenu)
SelectedIndex:Integer	setSelectedIndex(Integer)
SelectionMode:Integer	setSelectionMode(Integer)
TabStop:Boolean	setTabStop(Boolean)
ViewLink:Array	setViewLink(Array)
Additional Get Method	Additional Set Method
	clearSelection()
getFieldPath:String	
getNextSelectedIndex:Integer	
getNextSelectedIndex(fromIndex:Integer):Integer	
	resetSelectedIndex(index:Integer):Boolean
	resetSelectionInterval(StartIndex:Integer, StopIndex:Integer):Boolean
	setFieldPath(Value:String)

setSelectionInterval(StartIndex:Integer, StopIndex:Integer)

setFieldPath:String Array

This is the path from occurring view to field when linked to an occurring view. It specifies the path in the hierarchy from the view to a specific field.

ViewLink:Array

This is the link for the list when linked to an occurring view. For a list box, it specifies the occurring view that contains the data to be displayed.

Additional Action Methods

This object has methods related to focus. The *hasFocus()* method determines if the object currently has focus. The *setFocus()* method requests that focus be set to the object.

To select a single row, use the *setSelectedIndex()* method to specify the row you want to select. To select a contiguous range of rows, use the *setSelectionInterval()* method. Two methods are used to query the currently selected item. The *selectedIndex()* method returns the index of the currently selected item. The *nextSelectedIndex()* method returns the selected item following the one specified by the Index parameter.

Events

The *ListBox* object triggers the following events:

- [Click](#)
- [DoubleClick](#)
- [FieldError](#)
- [FocusGained](#)
- [FocusLost](#)



Not supported for thin client.

The *ListBox* object triggers the [Click](#) event when the selection is changed either by a mouse click or with the up and down arrow keys. It also triggers the [DoubleClick](#) event.

This object triggers the [FocusGained](#) event when the following occurs:

- The user tabs into the object or clicks on the object.

This object triggers the [FocusLost](#) event when the following occurs:

- The user tabs out of the object or clicks on another object in the window or on the window itself.

Menu

The *Menu* object displays choices on a menu bar and within a drop-down or popup menu containing additional selections. The following figure shows a sample menu dialog.

Sample menu



Constructor and Parameters

To create a Menu with pull-down items:

Create a MenuBar and add it to the Window using the *setMenuBar* method.

1. Create a Menu and add it to the MenuBar using the *add* method.

2. Create each MenuItem and add each one to the Menu using the *add* method.

For example:

```
dcl
  aMenuBar object type MenuBar;
  aMenu object type Menu;
  aChild object type MenuItem;
enddcl

proc InitProc for Initialize object TEST_WINDOW
(e object type InitializeEvent)
  map new MenuBar to aMenuBar
  map new Menu to aMenu
  set aMenu.text := 'File'
  aMenu.setMnemonic('F')
  map new MenuItem to aChild
  set aChild.text := 'Open'
  aChild.setMnemonic('O')
  TEST_WINDOW.setMenuBar(aMenuBar)
  aMenuBar.add(aMenu)
  aMenu.add(aChild)
endproc
```

There are no parameters for this object.

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object.

Menu object properties and methods

Property:Type (Get Method)	Set Method
Accelerator:Accelerator	setAccelerator(Accelerator)
Checked:Boolean	setChecked(Boolean)
CheckMandatoryFields:Boolean	setCheckMandatoryFields(Boolean)
Count:Integer	(read only)
IgnoreValidation:Boolean	setIgnoreValidation(Boolean)
Mnemonic:Char	setMnemonic(Char)
MnemonicKeycode:Integer	setMnemonicKeycode(Integer)
Selected:Boolean	setSelected(Boolean)
Text:String	setText(String)
Style:Integer	(Read only)
Validation:Boolean	setValidation(Boolean)
	Additional Action Method
	add(Item:MenuItem)
	addSeparator()
	add(Menu)
	getItem(integer):guiObject
	getItemCount()Integer

Additional Action Methods

The *Menu* object has two additional methods. The *add()* method appends MenuItems to the menu. The *addSeparator()* method appends separators.

Events

The *Menu* object triggers the following events:

- [Click](#)

MenuBar

The *MenuBar* object displays a menubar as part of the [Menu](#) object. In Java, *MenuBar* is not a GuiObject but a container for Menus and MenuItems. In C#, MenuBar and MenuItem are the same, therefore MenuBar is GuiObject as well.

Constructor and Parameters

The following is a sample declaration and construction:

```
dcl
  aMenuBar object type MenuBar;
enddcl
map NEW MenuBar() to aMenuBar
```

There are no parameters for this object.

Methods

The following methods are supported for this object:

```
bar.Add(Menu)
map bar.getMenu(index) to aMenu
map bar.getMenuCount() to anInt
```

Events

This object does not trigger any events.

MenuItem

The *MenuItem* object is used to display a menu choice as part of either the [Menu](#) object or the [PopupMenu](#) object. Menu items can be associated with a key combination that triggers the action associated with the menu item when pressed. The key combination is called an *accelerator* and is specified using the Accelerator property.

Constructor and Parameters

The following is a sample declaration and construction:

```
dcl
  aMenuItem object type MenuItem;
enddcl
map NEW MenuItem() to aMenuItem
```

There are no parameters for this object.

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object.

MenuItem object properties and methods

Property:Type (Get Method)	Set Method
Accelerator:Accelerator	setAccelerator(Accelerator)
Checked:Boolean	setChecked(Boolean)
CheckMandatoryFields:Boolean	setCheckMandatoryFields(Boolean)
Count:Integer	(read only)
IgnoreValidation:Boolean	setIgnoreValidation(Boolean)
Mnemonic:Char	setMnemonic(Char)
MnemonicKeycode:Integer	setMnemonicKeycode(Integer)
Selected:Boolean	setSelected(Boolean)
Text:String	setText(String)
Style:Integer	(Read only)
Validation:Boolean	setValidation(Boolean)
	Additional Action Method
	add(Item:MenuItem)
	addSeparator()
	getItem(integer):guiObject
	getItemCount()Integer

Additional Action Methods

The *MenuItem* object has two additional methods. The *add()* method appends MenuItems to the menu item. The *addSeparator()* method appends separators.

Accelerator:Accelerator

This property specifies a key combination that triggers a menu item's [Click](#) event without navigating the menu hierarchy, also referred to as *hot keys*. Valid values may be specified in the [Constants](#) class or the [Accelerator](#) class.

Checked:Boolean

This property specifies whether a menu item is displayed with a check mark. This property is equivalent to the [Selected:Boolean](#) property. It is provided for backwards compatibility with previous versions of the product.

For backward compatibility, when a checked MenuItem is clicked it will not toggle the checkmark but fires a clickevent (in Java, checked MenuItems will toggle when clicked). You must write code to uncheck the MenuItem.

A new appbuilder.ini setting is added to support the java functionality under the section [COMPATIBILITY], TOGGLE_CHECK_MENU_ONCLICK. The value for the ini key should be TRUE to support java functionality.

Style:Integer

This is a read-only property.
Returns one of the following values:

- Constants.PLAIN_MENUITEM
- Constants.CHECKBOX_MENUITEM

Events

The following event is triggered by this object:

- [Click](#)

MenuItem triggers the [Click](#) event when the user selects the menu item or when the user presses the accelerator key combination associated with that menu item.

MultiLineEdit

The *MultiLineEdit* object displays the standard multi-line edit field object, a rectangular area into which multiple lines of text can be entered. If the text is too long to display within the rectangular area, horizontal scrolling is automatically enabled. For a single-line edit field, refer to the [EditField](#) object.

When text is entered but not committed to the data link, it can be rolled back by pressing the Escape (*Esc*). Pressing *Enter* while focus is on an edit field with modified contents causes the changes to be committed to the data link. Moving focus away from the field also causes the changes to be committed.

MultiLineEdit has two methods. The *hasFocus()* method is used to determine if the object currently has focus. The *setFocus()* method is used to request that focus be set to the object.

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object.

MultiLineEdit object properties and methods

Property:Type (Get Method)	Set Method
Altered:Boolean	setAltered(Boolean)
AutoSelect:Boolean	setAutoSelect(Boolean)
DataLink:DataObject	setDataLink(DataObject)
Editable:Boolean	setEditable(Boolean)
EditLimit:Integer	setEditLimit(Integer)
Empty:Boolean	(read only)
Error:Boolean	(read only)
ImmediateReturn:Boolean	setImmediateReturn(Boolean)
InsertBreak:Boolean	setInsertBreak(Boolean)
InsertTab:Boolean	setInsertTab(Boolean)
Mandatory:Boolean	setMandatory(Boolean)
PopupMenu:PopupMenu	setPopupMenu(PopupMenu)
TabStop:Boolean	setTabStop(Boolean)
Text:String	setText(String)
WordWrap:Boolean	setWordWrap(Boolean)
Additional Get Method	
	setTabStop: Boolean

DataLink:DataObject

This is the link for the edit area of the multi-line edit. This property specifies the data field in the application hierarchy to which the edit field is linked.

The DataObject for a multi-line edit is *String* .

InsertBreak:Boolean

This property specifies whether pressing *Enter* in a multi-line edit field inserts a line break into the text or triggers the default push button.

InsertTab: Boolean

This property specifies whether pressing the **Tab** key in a multi-line edit field inserts a tab character into the text or transfers focus to another control. If *InsertTab* is *True* , focus can still be transferred with **Ctrl+Tab** .

WordWrap: Boolean

For multi-line edit fields, this property specifies whether the text is wrapped when it reaches the right edge of the edit area. The default value is *True* .

If the *WordWrap* property is set to *True* , text automatically wraps when it reaches the right edge of the edit area. If *InsertBreak* is set to *True* , pressing **Enter** causes a line break character to be entered into the text. If you want the user to be able to activate the default push button when focus is on the multi-line edit, set *InsertBreak* to *False* .

If *InsertTab* is *True* , pressing **Tab** causes the tab character or several space characters to be inserted in the text. If *InsertTab* is *False* , pressing **Tab** transfers focus to the next object. If *InsertTab* is *True* , **Ctrl+Tab** can be used to transfer focus.

Additional Action Methods

This object has methods related to focus. The *hasFocus()* method is used to determine if the object currently has focus. The *setFocus()* method is used to request that focus be set to the object.

Events

The *MultiLineEdit* object triggers the following events:

- [Click](#)
- [DoubleClick](#)
- [FieldError](#)
- [FieldValidation](#)
- [FocusGained](#)
- [FocusLost](#)



Not supported for thin client applications.

This object triggers the [Click](#) event when any of the following occur:

- The user presses the mouse button while the mouse is on the object.
- The user presses the object's mnemonic key (Alt+mnemonic key).
- The [Selected: Boolean](#) property is changed.
- A value is mapped into the field data-linked to the object.

This object triggers the [FocusGained](#) event when the following occurs:

- The user tabs into the object or clicks on the object.

This object triggers the [FocusLost](#) event when the following occurs:

- The user tabs out of the object or clicks on another object in the window or on the window itself.

MultiLineEdit provides for data validation using the [FieldError](#) and [FieldValidation](#) events. When an attempt is made to commit the data either by moving focus or pressing **Enter** , and the data contains errors, the [FieldError](#) event is triggered. If there are no errors, the [FieldValidation](#) event is triggered, allowing the application to verify that the data is acceptable.

PasswordField

The *PasswordField* object is very similar to the [EditField](#) object. The difference is that a password field displays only asterisks regardless of the data in the field. A Password field is implemented as a separate *PasswordField* object in the Java client to take advantage of the extra security provided by Java for password fields.

For additional information, see the section on [EditField](#).


Constructor and Parameters

The following is a sample declaration and construction:

```
dcl
  aPasswordField object type PasswordField;
enddcl
map NEW PasswordField() to aEditField
```

There are no parameters for this object.

Properties and Methods

 Inherits [GuiObject](#) and [EditField](#) and exposes all their properties and methods.

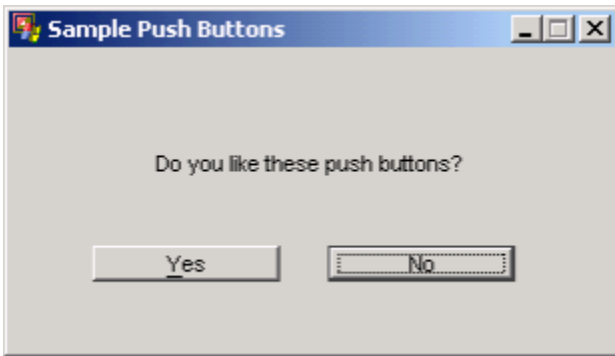
Events

The *PasswordField* object triggers the same events as the [EditField](#) object.

PushButton

The *PushButton* object is used to display standard push-button functionality. The following figure shows a sample push-button dialog.

Sample push buttons




Constructor and Parameters

The following is a sample declaration and construction:

```
dcl
  aPushButton object type PushButton;
enddcl
map NEW PushButton() to aPushButton
```

There are no parameters for this object.

Properties and Methods


 Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object.

PushButton object properties and methods

Property:Type (Get Method)	Set Method
CheckMandatoryFields:Boolean	setCheckMandatoryFields(Boolean)
HorizontalTextPosition:Integer	setHorizontalTextPosition(Integer)
IgnoreValidation:Boolean	setIgnoreValidation(Boolean)
Image:String	setImage(String)
Mnemonic:Char	setMnemonic(Char)

MnemonicKeyCode:Integer	setMnemonicKeyCode(Integer)
Pressed:Boolean	(read only)
PopupMenu:PopupMenu	setPopupMenu(PopupMenu)
TabStop:Boolean	setTabStop(Boolean)
Text:String	setText(String)
Validation:Boolean	setValidation(Boolean)
VerticalTextPosition:Integer	setVerticalTextPosition(Integer)
	Additional Set Method
	setVerticalTextToImagePosition:Integer


 getTabStop is not supported in C#.

VerticalTextPosition:Integer

This specifies the vertical position of the text.

HorizontalTextPosition:Integer

This specifies the horizontal position of the text.

 The VerticalTextPosition and HorizontalTextPosition methods and properties rely on a bitmap. There is no way to associate a bitmap with a label object. Hence these methods will not work.

Events

The *PushButton* object triggers the following events:

- [Click](#)
- [FocusGained](#)
- [FocusLost](#)

This object triggers the [Click](#) event when any of the following occur:


- The user presses the mouse button while the mouse is on the object.
- The spacebar is pressed while the object has the focus.
- The user presses the object's mnemonic key (Alt+mnemonic key).

This object triggers the [FocusGained](#) event when the following occurs:

- The user tabs into the object or clicks on the object.

This object triggers the [FocusLost](#) event when the following occurs:

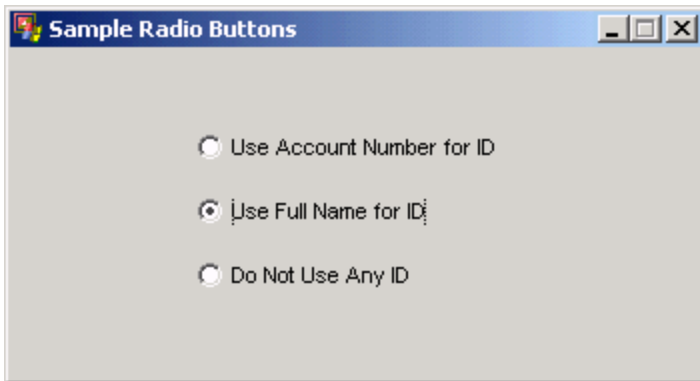
- The user tabs out of the object or clicks on another object in the window or on the window itself.

 FocusLost and FocusGained events are not supported for thin client applications.

RadioButton

The *RadioButton* object is used to display the standard radio button functionality. You can also use the [CheckBox](#) object for turning a particular setting either on or off.

Sample radio buttons



Constructor and Parameters

The following is a sample declaration and construction:

```

dcl
  aRadioButton object type RadioButton;
enddcl
map NEW RadioButton() to aRadioButton

```

There are no parameters for this object.



You cannot select a radio button on an HTML servlet if the radio button does not have a link to a field.

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object.

RadioButton object properties and methods

Property:Type (Get Method)	Set Method
Altered: Boolean	setAltered(Boolean)
Data Link: DataObject	setDataLink(DataObject)
ImmediateReturn: Boolean	setImmediateReturn(Boolean)
Mnemonic: Char	setMnemonic(Char)
MnemonicKeycode: Integer	setMnemonicKeycode(Integer)
PopupMenu: PopupMenu	setPopupMenu(PopupMenu)
Selected: Boolean	setSelected(Boolean)
TabStop: Boolean	setTabStop(Boolean)
Text: String	setText(String)

DataLink: DataObject

Radio buttons are linked to character (string) fields. When a radio button is selected, the system identifier (HPSID) for the button is placed in the data-linked character field.

The DataObject for a radio button is String.

Additional Action Methods

This object has methods related to focus. The `hasFocus()` method is used to determine if the object currently has focus. The `setFocus()` method is used to request that focus be set to the object.

Events

The `RadioButton` object triggers the following events:

- [Click](#)
- [DoubleClick](#)
- [FieldError](#)
- [FocusGained](#)
- [FocusLost](#)

This object triggers the [Click](#) event when any of the following occur:


- The user presses the primary mouse button when the mouse is on the object.
- The spacebar is pressed when the object has the focus.
- The user presses the object's mnemonic key (that is, Alt+mnemonic key).
- The [Selected:Boolean](#) property is set to True.
- A value is mapped into the field data-linked to the object.

This object triggers the [FocusGained](#) event when any of the following occur:

- The user tabs into the object or clicks on the object.

This object triggers the [FocusLost](#) event when any of the following occur:

- The user tabs out of the object or clicks on another object in the window or the window itself.

 These events are not supported for thin client.

`RadioButton` provides for data validation using the [FieldError](#) event. When an attempt is made to commit the data either by moving focus or pressing `Enter`, and the data contains errors, the [FieldError](#) event is triggered.

Rectangle

A `Rectangle` object is used to display a rectangle or square.

Support


 This is *not* supported for thin (HTML) clients.

Constructor and Parameters

There are no parameters for this object. The following is a sample declaration and construction:

```
dcl
  aRectangle object type Rectangle;
enddcl
map NEW Rectangle() to aRectangle
```

Properties and Methods

 Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object.

Rectangle object properties and methods

The `Rectangle` class contains no properties or methods except the properties or methods of the parent class.

Events

- [Click](#)
- [DoubleClick](#)

This object triggers the [Click](#) event when any of the following occur:

- The user presses the primary mouse button when the mouse is on the object.

TabControl

The *TabControl* object allows user to group controls into multiple pages and to switch between these tab pages. A tab control can be created statically by using Window Painter, or can be created dynamically at runtime. To create dynamically a tab control, it is created a *TabControl* object, then individual tab pages are added using the *addPage()* method.

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object.

TabControl object properties and methods

Property	Set Method
Count:Integer	(read only)
MultipleRows:Boolean	setMultipleRows(flag:Boolean)
Orientation:Integer	setOrientation(value:Integer)
SelectedIndex:Integer	
SelectedTab:TabPage	
TabStop:Boolean	setTabStop(Boolean)
Additional Get Method	Additional Set Method
	addPage(page:TabPage)
getPage(index:Integer):TabPage	
	insertPage(index:Integer, page:TabPage)
	removePage(page:TabPage)
	removeAt(index:Integer)
	setEnabledPage(index:Integer, value:Boolean)

addPage(page:TabPage)

Adds a tab page to the tabcontrol.

getPage(index:Integer):TabPage

Gets the TabPage at the given index.

insertPage(index:Integer, page:TabPage)

Adds a tab page to the tabcontrol at the given index.

removePage(page:TabPage)

Removes a tab page from the tabcontrol.

removeAt(index:Integer)

Removes a tab page from the tabcontrol at the given index.

setEnabledPage(index:Integer, flag:Boolean)

Enables or disables a tab page at the given index.

setMultipleRows(flag:Boolean)

If the flag is true, tab pages will be shown in multiple rows, otherwise it will be shown in a scrollable single row of tabpages.

setOrientation(value:Integer)

Sets the orientation of the tab pages to TOP or BOTTOM.

Events

The *TabControl* object triggers the following events:

- TABPAGE Deselected
- TABPAGE Selected

Table

The *Table* object is used to display a standard table with multiple rows and columns. In the Window Painter (and in previous versions of this product), it is referred to as a multicolumn list box (MCLB) or spreadsheet. The *Table* object supports the concept of virtual rows. That is, it can be associated with a data source that contains more rows than can be displayed at one time. Thus, the row numbers reflect the row numbers in the database rather than the actual row numbers on the displayed table.

Sample table with three columns

ID	Name	Project
6	ZGYBTZ8	APPB_REP_SRV
7	ZGZOJZ8	APPB_RULE_DTL_ACT
8	ZGZNAZ8	APPB_RULE_DTL_DIS
9	ZGZOKZ8	APPB_RULE_DTL_INQ
10	ZGZPBZ8	APPB_SS_LB_DIS
11	ZGYBPZ8	APPB_SS_MCLB_ACT
12	ZGYAZZ8	APPB_SS_MCLB_DIS
13	ZGYBQZ8	APPB_SS_MCLB_INQ
14	ZGZFPZ8	APPB_SS_MCLB_VAL
15	ZGZMIZ8	APPB_STD_DT_FROM_CHAR

When using ObjectSpeak to manipulate tables, tables are implemented using both [Table](#) and [Column](#) objects. In general, a *Table* object possesses one or more [Column](#) objects. The *Table* class has a number of properties that affect the table as a whole. The *Column* class contains a number of properties that determine the appearance and behavior of individual columns.

You can set properties or methods on the table or on individual columns of the table. As with all other visual objects, the names of tables and columns are the same as their system identifier (HPSID).

Use these steps to dynamically construct a table:

***Create a Table object.**

1. Create one or more [Column](#) objects.
2. Add them to the Table using the Table's addColumn() method.

This method appends the specified Column object to the right of the table. For tables created in the Window Painter, the Column objects are automatically created and added to the table by the generated Java code.

By default, the background color of Numbering column and Header is the scrollbar's color. The default background and foreground colors of the Header can be overridden using the setHeaderForeground(Color) and setHeaderBackground(Color) methods.

For information about scrolling, refer to [Smooth Scrolling in a Table Object](#).

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists the properties and methods for this object.

<i>Table object properties and methods</i>	
Property:Type (Get Method)	Set Method
Altered:Boolean	setAltered(Boolean)
AutoSelect:Boolean	setAutoSelect(Boolean)
BackBuffer:Integer	setBackBuffer(Integer)
BackGrndColor:Color	
Border:Integer	
BorderStyle:Integer	setBorderStyle(Integer)
CurrentColumn:Integer	
CurrentRow:Integer	
DatabaseSize:Integer	(read only, set only through (Integer, Integer))
Editable:Boolean	setEditable(Boolean)
ElevatorPosition:Integer	(read only)
Empty:Boolean	(read only)
Error:Boolean	(read only)
FirstVisibleRow:Integer	setFirstVisibleRow(Integer)
ForeGrndColor:Color	
HeaderBackground:Color	setHeaderBackground(Color)
HeaderFont:Font	setHeaderFont(Font)
HeaderForeground:Color	setHeaderForeground(Color)
HeaderHeight:Integer	setHeaderHeight(Integer)
ImmediateReturn:Boolean	setImmediateReturn(Boolean)
Justification:Integer	setJustification(Integer)
LastVisibleRow:Integer	setLastVisibleRow(Integer)
Lines:Integer	setLines(integer)
Mandatory:Boolean	setMandatory(Boolean)
NextSelectedIndex:Integer	(read only)
NumberingColumn:Boolean	setNumberingColumn(Boolean)
PopupMenu:PopupMenu	setPopupMenu(PopupMenu)
RowHeight:Integer	setRowHeight(Integer)
RowSelect:Boolean	setRowSelect(Boolean)
ScrollableOccurs:Integer	(read only)
ScrollLock:Boolean	setScrollLock(Boolean)
ScrollBars:Integer	setScrollBars(Integer)

SelectedIndex:Integer	setSelectedIndex(Integer)
SelectedRowCount:Integer	
SelectionMode:Integer	setSelectionMode(Integer)
TabStop:Boolean	setTabStop(Boolean)
ViewLink:Array	setViewLink(Array)
	setVirtualListBoxSize (read-only)
VisibleOccurs:Integer	(read only)
Additional Get Method	Additional Set Method
	addColumn(ColObj:Column)
	clearSelection()
convertToPhysical(Integer):Integer	
	disableTopAndBottomEvents(Boolean)
getColumn(index:Integer):Column	
getColumnCount():Integer	
getFirstVisibleOccurrence():Integer	
getLastVisibleOccurrence():Integer	
getListLink():Array	
getNextSelectedPhysicalIndex():Integer	
getNextSelectedIndex(fromIndex:Integer):Integer	
getNextSelectedIndex():Integer	
getOccurs():Integer	
getScaledHeaderHeight():Integer	
getScaledRowHeight():Integer	
getSelectedPhysicalIndex():Integer	
getSelectedVirtualIndex():Integer	
getVisibleRows():Integer	
isAutoSelect:Boolean	
isRowSelect:Boolean	
nextSelectedIndex(fromIndex:Integer):Integer	
	resetSelectedIndex():Integer
	resetSelectionInterval(StartIndex:Integer, StopIndex:Integer)
	setListLink(Array)
	setMoreData(Boolean)
	setMoreRows(n:Integer, flag:Boolean)
	setScaledHeaderHeight(Integer)
	setScaledRowHeight(Integer)
	setSelectionInterval(StartIndex:Integer, StopIndex:Integer)
	setVirtualListBoxSize(TopVirtualRow:Integer, VirtualTableSize:Integer)
	setStyleClass(cssClassName:String)

	setCellStyleClass(row:Integer, col:Integer, cssClassName:String)
	setRowStyleClass(row:Integer, cssClassName:String)
	setColumnStyleClass(col:Integer, cssClassName:String)

The table itself is data-linked to an occurring view and each column in the table (other than the optional numbering column) is associated with a field in the hierarchy beneath the occurring view. The complete data link consists of a data link from the table to the occurring view and for each column, a specification of the path from the occurring view to the particular field in the view to which the column is linked. These data-links are specified using the set methods with the following properties:

- `<tablename >.ViewLink` ? link to an occurring view
- `<columnname >.FieldPath` ? path from an occurring view to a field

The first is set with the [Table](#) object. The second is set with [Column](#) object.

FirstVisibleRow:Integer

This property specifies the first visible row in the table.

getColumn(index:Integer):Column

This method returns the column at the given index or null if not found (index in 1 based index).

getColumnCount():Integer

This method returns the number of columns in the table.

HeaderBackground:Color

This property specifies the color of the background of the header row of the table.

HeaderForeground:Color

This property specifies the color of the foreground of the header row of the table.

HeaderHeight:Integer

This property specifies the height, in pixels, of the header row of the table.

LastVisibleRow:Integer

This property specifies the last visible row in the table.

Lines:Integer

This property specifies the horizontal and vertical lines in the table.
The values for this property can be one of the following:

- Constants.NO_LINES
- Constants.HORIZONTAL_LINES
- Constants.VERTICAL_LINES
- Constants.HORIZONTAL_AND_VERTICAL_LINES

NumberingColumn:Boolean

This property specifies whether a column should be automatically added in the left-most position of a table to display the (virtual) row number. The default maximum width of the numbering column is five times the average character width.

If *NumberingColumn* is *True* , then a column which contains (virtual) row numbers is added at the extreme left of the table. The absence or presence of the numbering column does not effect the index number of the remaining column, the left-most non-numbering column always has a column index of 1.

RowSelect:Boolean

This property specifies whether an entire table row is selected when a cell in that row is selected. If *RowSelect* is *True* , then clicking anywhere on a row causes the entire row to be selected; otherwise, only the cell that is clicked is selected.

SelectedRowCount:Integer

This is a read-only property. It returns an integer indicating the number of selected rows in a table (multicolumn list box), or 0 if none. For

example,

```
dcl
numSelectedRows integer;
enddcl
set numSelectedRows := {MCLB_HPSID}.SelectedRowCount
```



This is not supported in thin client applications.

setVirtualListBoxSize (read-only)

This method specifies the dimensions of a virtual list box. The method can be called in R/O mode too.

```
setStyleClass(cssClassName:String),
setCellStyleClass(row:Integer, col:Integer, cssClassName:String),
setRowStyleClass(row:Integer, cssClassName:String),
setColumnStyleClass(col:Integer, cssClassName:String)
```

These methods specify a css class for a table, a table cell, a table row and a table column. The method can be called in HTML mode only.

ViewLink:Array

This is the link for the Table when linked to an occurring view. For a Table, this property specifies the occurring view that contains the data to be displayed.

Additional Action Methods

Use the `addColumn()` method when constructing a Table to append the specified Column object to the right of the table. When the `disableTopAndBottomEvents` method is called with a value of `True`, HPS_LB_BOTTOM and HPS_LB_TOP events are not initiated. The `setMoreRows` method is used to simplify incremental smooth scrolling. In this case the actual database size is not known, so smooth scrolling is activated by incrementing the virtual size of the table by n rows on every [DataRequired](#) event. If this method is called with `setMoreRows(1, True)`, on every [DataRequired](#) event, the virtual size of the table is incremented by 1 row more than the current virtual bottom limit. This enables smooth scrolling. When a fetch returns less than the occurring size of rows, the scroll down event can be disabled by calling `setMoreRows(remaining rows, False)`. If dynamic views are not used, the incremental scrolling is re-enabled when scrolling *up*, past the virtual top limit. For information on scrolling, refer to [Smooth Scrolling in a Table Object](#). The `SelectedIndex` methods are used to query the currently-selected rows or cells. The `getSelectedIndex()` method returns the index of the currently selected row. To select a single row, use `setSelectedIndex()` to specify the row you want to select. To select a contiguous range of rows, use the `setSelectionInterval()` method. Currently these methods refer to the display row number, where the topmost displayed row has an index of 1. Use the `clearSelection()` method to clear the selection. For a description of the `getElevatorPosition()`, `setFirstVisibleRows()`, `setLastVisibleRows()`, and `setVirtualListBoxSize()` methods, see [Smooth Scrolling in a Table Object](#).

Events

The `Table` object triggers the following events:

- [CellClick](#)
- [CellDoubleClick](#)
- [CellFocusGained](#)
- [CellFocusLost](#)
- [Click](#)
- [DataRequired](#)
- [DoubleClick](#)
- [EnterKeyPressed](#)
- [FieldError](#)
- [FieldValidation](#)
- [HeaderClick](#)
- [RowFocusGained](#)
- [RowFocusLost](#)

The `Table` object triggers a number of events. When the table needs more data, the [DataRequired](#) event is triggered. When a cell gains focus, the [CellFocusGained](#) event is triggered; when it loses focus, the [CellFocusLost](#) event is triggered.

The `Table` object also triggers the [Click](#) and [DoubleClick](#) events when the user clicks or double-clicks on the table. The parameters of the event provide information about the row and column where the click or double-click occurred.

When the contents of a cell are changed and focus is shifted away from the cell or **Enter** is pressed while the focus is in the changed cell, field-level validation occurs. If the user has entered syntactically erroneous data in the field, the [FieldError](#) event is triggered. Otherwise, the [FieldValidation](#) event is triggered, allowing the application to validate the contents of the field.



This client only supports the DoubleClick event.

Smooth Scrolling in a Table Object

Several methods play a role in smooth scrolling in a *Table* object. Smooth scrolling refers to fetching data when needed from a data source and placing it in the occurring view to which the table and columns are linked. When a table needs data that is not already in the occurring view, it triggers the [DataRequired](#) event. The [DataRequired](#) event procedure calls `getElevatorPosition()` to determine what virtual row should be placed in the top of the occurring view. Once the data has been fetched and placed in the occurring view, then `setVirtualListBoxSize()` is called to specify which virtual row was actually placed in the top of the occurring view and the total number of virtual rows in the data source. The `setFirstVisibleRow()` method specifies that a virtual row, other than the one at the top of the occurring view, is the top displayed row. There are many different ways of smooth scrolling, for instance:

- **Count(*) method** ? The database size is known at the beginning, or the database count is done before the fetch to determine the actual database size. The advantage of this is that the table's scrollbar shows the actual database size; the disadvantage is that the database count is expensive.
- **Incremental method** ? The database size is not known, so set the virtual size of the table to either the current size plus the increment after each fetch (if the database has more data than displayed in the table) or the database size when no more data to be fetched. The disadvantage is that the table's scrollbar never shows the actual database size.
- **Dynamic views** ? Another method is to combine dynamic views with one of the other two methods. The advantage is that the table grows every time fetch and append is done, and scrolling backward (up) does not generate a data-required event.

For more details and examples, see:

- [Incremental Smooth Scrolling with setMoreRows](#)
- Incremental Smooth Scrolling with Dynamic Views

Incremental Smooth Scrolling with setMoreRows

Incremental scrolling can be simplified by using the `setMoreRows(n,flag)` method. For a detailed description, refer to the `setMoreRows` method topic in [Additional Action Methods](#). The following sample code demonstrates incremental smooth scrolling using this method.

- [List Display Rule](#)
- [Fetch Rule](#)

List Display Rule

```
*>-----
Rule : AB_SMOOTH_SCROLL_INC_DIS - list display rule
Version : AppBuilder 3.1

This Rule demonstrates smooth scrolling using <mclb>.setMoreRows(x, flag) and <mclb>.setBackBuffer(n)
for INCREMENTAL scrolling.
For INCREMENTAL scrolling, the database size is not known (count(*) not allowed), so the fetch rule
uses a variable to identify if more data are available after the fetch;
initially call setMoreRows(increment, true) to enable smooth scrolling and on the last fetch call
setMoreRows(remaining rows, false) to disable scroll DOWN events.

setBackBuffer(n)
  where n: number of rows to fetch backward

setMoreRows(x, flag)
  where x: is integer and denotes increment or number of rows in the last fetch,
  flag: when true, the virtual size is incremented x times on every DataRequiredEvent and when false
  virtual size is set to virtual top + x - 1 to disable further scrolling down

Steps for using setMoreRows(rows,type)
for incremental scrolling
-----
1. call the method
  <mclb>.setBackBuffer(x)
2. Fetch next block and map
  if more data available then call the method
  <mclb>.setMoreRows(1, true)
  else call the method
  <mclb>.setMoreRows(x, false)
3. On Every DataRequiredEvent
```

- a. Get Next block and append
- b. On the last fetch call the method setMoreRows(n, false) where n is the remaining rows

```
-----<*

dcl

BackBufferAmt smallint;
enddcl

//-----
// Fetch the database for the next block of data and
// map it to the occurring view of MCLB.
// The fetch rule returns -1 for more data available than the fetch
// size or the remaining number of rows if it is the last block.
// 1. Get data from database from index in fromindex.
// 2. Map to MCLB occurring view.
// 3. If the AB_MORE_DATA > 0
// stop scroll down with setMoreRows(AB_MORE_DATA, false)
//-----
proc GetNextBlock(fromIndex smallint)
//Fetch next block from the index from index
map fromIndex
to AB_ELEV_POS of AB_MERULE_NOCOUNT_SQL_FET_I
use rule AB_MERULE_NOCOUNT_SQL_FET
//Map fetch out view to occurring view of MCLB
map AB_MERULE_D of AB_MERULE_NOCOUNT_SQL_FET_O
to AB_SMOOTH_SCROLL_INC_OCC of AB_SMOOTH_SCROLL_INC_W
if AB_MORE_DATA of AB_MERULE_NOCOUNT_SQL_FET_O > 0
//Touched database bottom
//No more rows DOWNwards
MERULE_MCLB.setMoreRows(AB_MORE_DATA of
AB_MERULE_NOCOUNT_SQL_FET_O,
false)
endif
endproc

//-----
// This procedure gets called when a window is initialized
// the first time before shown.
// Here:
// 1. Call setMoreRows(inc, true) to start smooth scrolling.
// 2. Set BackBuffer amount.
// 3. Call getNextBlock to get first block.
//-----
proc InitEventProc for initialize object AB_SMOOTH_SCROLL_INC
(e object pointer to InitializeEvent)
// Starts incremental scrolling incr=1
MERULE_MCLB.setMoreRows(1, true)
// 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 does not have to fetch data
// outside of his view.
map (MERULE_MCLB.ScrollableOccurs - MERULE_MCLB.VisibleOccurs) / 2
to BackBufferAmt
MERULE_MCLB.setBackBuffer(BackBufferAmt)
// Fetch next block of data and map
GetNextBlock(1)
endproc

//-----
// This procedure is called every time user scrolls pass
// the current virtual limit.
// Here we
// 1. If e.TypeString = 'OutOfRange' Fetch,
// and map next block of data.
//-----
proc DataEventProc for DataRequired object MERULE_MCLB
(e object pointer to DataRequiredEvent)
if(e.TypeString = 'OutOfRange')
GetNextBlock(e.TopVirtualRow)
endif
endproc
```

```
//-----  
// Click Event Procedure for the Pushbutton EXIT.  
// Terminate the window here.  
//-----  
proc ExitProc for Click object EXIT  
(e object pointer to ClickEvent)
```

```
AB_SMOOTH_SCROLL_INC.terminate
endproc
```

Fetch Rule

```
*>
Rule :AB_MERULE_INC_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.
Fetch by name.
-----<*
```

```
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_INC_SQL_FET_I
to L_SEARCH_FIELD
map occurs(AB_MERULE_D)
to L_OCCURSIZE
```

```
*> Select all records meeting search criteria into cursor <*
```

```
sql asis
declare MERULE1 cursor for
select A.SHORTNAME,
A.NAME,
A.REMOTEMAINTEANCED,
A.REMOTEMAINTEAINEDBY,
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_INC_SQL_FET_O
return
endif
```

```
*> Fill the occurring 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_INC_SQL_FET_O(L_COUNT)
enddo
```



```
close MERULE1
endsql
```



For an example of smooth scrolling using system components, refer to the section on smooth scrolling in the *System Components Reference Guide*.

TabPage

TabPage is the container for the *GuiObjects* that can be added into a tab control. A tab page cannot be displayed directly into a window, it should be the part of a tab control *GuiObjects* can be added into the tab page using the *add()* method.

Property:Type	Set Method
DisabledImage:String	setDisabledImage(String)
Image:String	setImage(String)
Title:String	setTitle(String)
Additional Get Method	Additional Set Method
	addImage(String)
	setHpsId(String)

DisabledImage:String

Sets or gets the image for the tab page when it is disabled.

Image(String name)

Sets or gets the image for the tab page.

Title(String)

Sets or gets the title of the tab.

addChild(GuiObject)

Adds a *GuiObject* into the tab page.

Dynamic-Only Control Objects

Dynamic-Only Control Objects

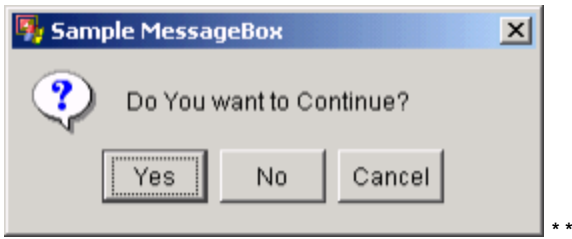
The controls for a user interface, that can be defined only during execution time using ObjectSpeak syntax in Rules source code (dynamic-only objects) are:

- [MessageBox](#)
- [PopupMenu](#)
- [Timer](#)
- [TreeView](#)
- [TreeNode](#)

MessageBox

The *MessageBox* object is used to display a message box on the screen. To use a message box, an instance of *MessageBox* must first be created using the new operator, as illustrated below. Then properties are set to specify the message, title, buttons, and icon. Finally, the *showMessageBox()* method is called to display the message box. When the user closes the message box, the *Show* method returns. If there is more than one button, the return value of *showMessageBox()* indicates which button was pressed.

Sample message box



Constructor and Parameters

The following is a sample declaration and construction:

```
dcl
    SaveMessageBox object type MessageBox;
    MessageBoxReturn Integer;
enddcl
map NEW MessageBox() to SaveMessageBox
```

There are no parameters for this object.

Properties and Methods

The following table lists the properties and methods for this object.

MessageBox object properties and methods

Property:Type (Get Method)	Set Method
Argument1:String	setArgument1(String)
Argument2:String	setArgument2(String)
Argument3:String	setArgument3(String)
ButtonType:Integer	setButtonType(Integer)
Locale:Locale	setLocale(Locale)
Message:String	setMessage(String)
MessageType:Integer	setMessageType(Integer)
Parent:Window	setParent(Window)
Title:String	setTitle(String)
	Additional Action Method
	show() :Integer

Argument1:String

This is the first string that can be substituted into the optional argument of a message in a message box.

Argument2:String

This is the second string that can be substituted into the optional argument of a message in a message box.

Argument3:String

This is the third string that can be substituted into the optional argument of a message in a message box.

ButtonType:Integer

This property specifies the button or buttons that are displayed in a message box. Valid values are defined in the [Constants](#) class as:

- `DEFAULT_BUTTONS`

- `OK_BUTTON`
- `OK_CANCEL`
- `YES_NO`
- `YES_NO_CANCEL`

For example, if the message box should have Yes, No, and Cancel buttons, set `ButtonType` to `YES_NO_CANCEL`, as shown in the code in the [Example: Message Box](#).

The `ButtonType` property is not supported for thin client applications.

Message:String

This property specifies the message to be displayed in a message box. Up to three substrings can be inserted into the message. The substrings are specified by the `Argument1`, `Argument2`, and `Argument3` properties. `Argument1` is substituted at the location of `%1` in the message, and similarly for the other arguments; this is illustrated in the [Example: Message Box](#).

MessageType:Integer

This property specifies the type of icon that is displayed in a message box. Valid values, as defined in the `Constants` class, are as follows:

- `ERROR`
- `INFORMATION`
- `PLAIN`
- `QUESTION`
- `WARNING`



Java thin clients support only `INFORMATION` and `QUESTION` message box types. These are the only types available when using message box functionality in JavaScript.

Parent:Window

This property specifies the parent window.

The `Parent` property is not supported for thin client applications.

Title:String

This property specifies the title (or caption) for a message box or window.

The `Title` property is not supported for thin client applications.

Additional Action Methods

If the message box has more than one button, then the application should use the return value of the `show()` method to decide what to do. The `showMessageBox()` method returns one of the following values (defined in [Constants](#)):

- `OK`
- `YES`
- `NO`
- `CANCEL`

An `INFORMATION` message box has an OK button.

A `QUESTION` message box has an OK and a Cancel buttons.

Thin Client Support

`Title`, `ButtonType`, and `Parent` properties are not supported for thin client.

Events

The `MessageBox` object does not trigger any events.

Example: Message Box

This example shows a declaration section that contains local variables needed for the code and the code to create, configure, and display the message box. It then shows the logic for responding to the user's choice. Include the code within either an event procedure or a standard procedure.


```

dcl
    SaveMessageBox object type MessageBox;
    MessageBoxReturn Integer;
enddcl
*> create a message box <*>
map new MessageBox to SaveMessageBox

*> set message box properties <*>
SaveMessageBox.SetMessageType(Constants.QUESTION)
SaveMessageBox.SetButtonType(Constants.YES_NO_CANCEL)
SaveMessageBox.SetTitle('Save File')
SaveMessageBox.SetMessage('Save the file named %1?')
SaveMessageBox.SetArgument1('SAMPLE.XML')

*> display message box <*>
map SaveMessageBox.Show to MessageBoxReturn

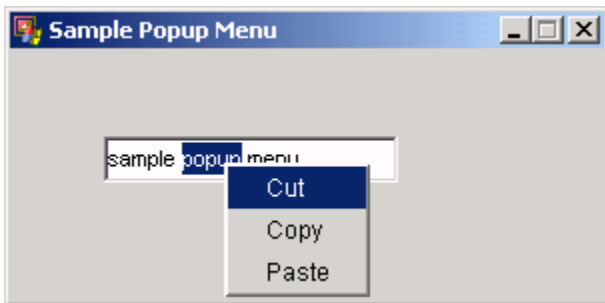
*> respond to user's choice <*>
CASEOF MessageBoxReturn
    CASE (Constants.YES)
        *> save the file and exit the application <*>
    CASE (Constants.NO)
        *> do not save the file, and exit the application <*>
    CASE (Constants.CANCEL)
        *> do not save the file, and return to the application <*>
ENDCASE

```

PopupMenu

The *PopupMenu* object is used to display a menu that appears when the secondary mouse button is clicked. This object does not generate any events but the menu items that are added to it trigger [Click](#) events when they are clicked.

Sample popup menu



The PopupMenu is not supported for thin client applications.

Properties and Methods

The following table lists the properties and methods for this object:

<i>PopupMenu object properties and methods</i>
Action Methods
add(Item:MenuItem)
addSeparator()

PopupMenu:PopupMenu

This method sets the popup menu that is displayed for objects on a window and the window itself, typically by clicking the right mouse button or secondary mouse button.

If a given user interface object does not have a popup menu, but the window does, right clicking on the object causes the window's popup menu to be displayed.

If you want only one popup menu for the window and all its objects, just define a popup for the window.

A PopupMenu can be added to nearly any of the user interface objects by using their PopupMenu property (or the setPopupMenu() method). A PopupMenu can also be added to the window itself.

Additional Action Methods

The add() method appends menu items to the popup menu. The addSeparator() method appends separators.

Example: Creating Popup Menus

The following code creates a popup menu that implements the standard Cut, Copy, and Paste operations. It declares local variables for the popup menu, as well as the menu items that appear on the menu. It then defines a procedure which builds the popup menu.

Note that as each menu item is created, an event procedure (or handler) for it is defined. The window Initialize Event procedure calls this procedure and then adds the popup menu to an edit field. Finally, the event procedure that actually handles click events generated from the popup menu is defined. The event procedure is defined in such a way that it handles events triggered from all the menu items in the application (since it specifies that it is for type MenuItem rather than for a particular menu item).

```

*> declare local variables <*>
dcl
    StandardPopupMenu object type PopupMenu;
    CutMenuItem,
    CopyMenuItem,
    PasteMenuItem object type MenuItem;
*> forward declare event procedure for popup menu <*>
    MenuClick proc for Click type MenuItem (e object type ClickEvent);
enddcl

*> procedure to build standard popup menu <*>
proc BuildPopupMenu
    *> create popup menu <*>
    map new PopupMenu to standardPopupMenu
    *> create Cut menu item <*>
    map new MenuItem to CutMenuItem
    CutMenuItem.setHpsID("CutItem")
    CutMenuItem.setText("Cut")
    CutMenuItem.setMnemonic('t')
    Handler CutMenuItem(MenuClick)

    *> create Copy menu item <*>
    map new MenuItem to CopyMenuItem
    CopyMenuItem.setHpsID("CopyItem")
    CopyMenuItem.setText("Copy")
    CopyMenuItem.setMnemonic('C')
    Handler CopyMenuItem(MenuClick)

    *> create Paste menu item <*>
    map new MenuItem to PasteMenuItem
    PasteMenuItem.setHpsID("PasteItem")
    PasteMenuItem.setText("Paste")
    PasteMenuItem.setMnemonic('P')
    Handler PasteMenuItem(MenuClick)

    *> add menu items to popup menu <*>
    standardPopupMenu.add(CutMenuItem)
    standardPopupMenu.add(CopyMenuItem)
    standardPopupMenu.add(PasteMenuItem)
endproc

*> window initialization event procedure <*>
proc MainWindowInitialize for Initialize object MAIN_WINDOW
(e object type InitializeEvent)
    *> create the popup menu <*>
    BuildPopupMenu

    *> assign the popup menu to some edit fields <*>
    NameField.SetPopupMenu(StandardPopupMenu)
    DateField.SetPopupMenu(StandardPopupMenu)
    TimeField.SetPopupMenu(StandardPopupMenu)
endproc

*> click event handler for MenuItem objects <*>
proc MenuClick for Click type MenuItem
(e object type ClickEvent)
    CASEOF e.HpsID
        CASE 'CutItem'
            *> cut the selected text <*>
        CASE 'CopyItem'
            *> copy the selected text <*>
        CASE 'PasteItem'
            *> paste text from clipboard <*>
    ENDCASE
endproc

```

Timer

The *Timer* object is used to notify the application once or repeatedly that a specified time has elapsed. The notification is in the form of a Timer event. *Timer*s are non-visual objects at runtime; that is, they do not appear on the window. *Timer*s are typically created dynamically at runtime, as shown in the sample code below.

Timers must be created with the new keyword and mapped to a local variable of type *Timer*, as illustrated in the code below. The name of the window must be passed to the *Timer* constructor method that follows the new keyword.



The *Timer* object is not supported for the thin client.

Properties and Methods

The following table lists the properties and methods for this object.

Timer object properties and methods

Property:Type (Get Method)	Set Method
Enabled:Boolean	
Visible:Boolean	setVisible(Boolean)
Delay:Integer	setDelay(Integer)
HpsID:String	setHpsID(String)
Repeats:Boolean	setRepeats(Boolean)
Running:Boolean	(see Note below table)
	Additional Action Methods
	start()
	stop()



Running is a read-only property and the set method is *not* supported.

Delay:Integer

This property specifies the time, in milliseconds, between successive *Timer* events generated by the *Timer* control.

Enabled

This is a read-only property and returns *TRUE* when *Timer* is enabled. *Timer* is enabled (or generates a *Timer* event) only when the window of the *Timer* is enabled.

Repeats:Boolean

This property specifies whether a timer triggers repeatedly or just once. By default, a timer triggers repeatedly.

Running:Boolean

This property specifies whether a timer is enabled and, therefore, running. This is a read-only property.

Additional Action Methods

The *start()* method starts the timer, and the *stop()* method stops it. If the timer triggers only a single event, there is no need to call *stop()*.

Events

The *Timer* object triggers the following event:

- [Timer](#)

The *Timer* object generates only the [Timer](#) event. To cause the timer to trigger just once, set the *Repeats* property to *False*. The time interval


between successive timer events is specified with the Delay property; units are in milliseconds. This also represents the time between when the start() method is called and when the first event is triggered. The time interval is approximate.

Example: Creating Event Procedures for Timers

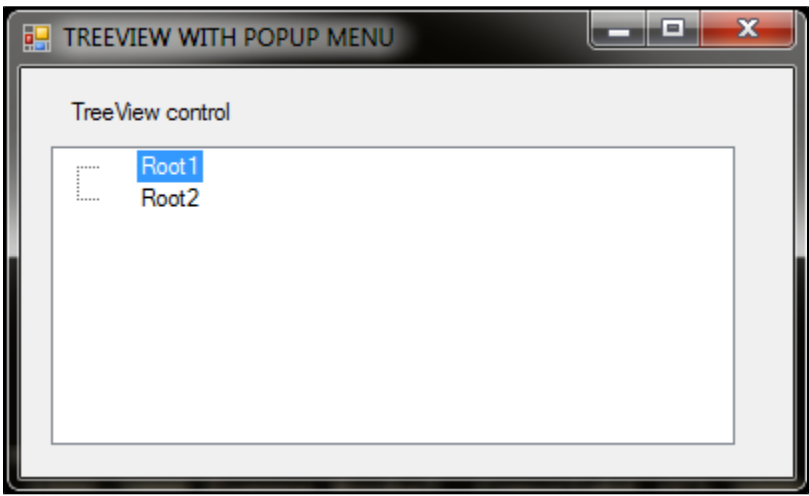
```
dcl
UpdateTimer object type Timer;
enddcl
*> event procedure for Timer events <*>
proc TimerProc for Timer object UpdateTimer
(e object type TimerEvent)
*> update whatever needs to be updated! <*>
*> add code here <*>
*> stop the timer when appropriate <*>
*> UpdateTimer.Stop <*>
endproc
*> respond to Update button click by starting timer <*>
proc UpdateButtonClick for Click object UpdateButton
(e object type ClickEvent)
*> create a timer <*>
map new Timer(MAIN_WINDOW) to UpdateTimer
*> set time interval to 1 second (1000 millisec) <*>
UpdateTimer.SetDelay(1000)
*> specify that timer should repeat until the Stop
method is called <*>
UpdateTimer.SetRepeats(True)
*> start the timer <*>
UpdateTimer.Start
*> dynamically add event procedure to handle
timer events <*>
Handler UpdateTimer(TimerProc)
endproc
```

TreeView

The *TreeView* object displays a hierarchical collection of labeled items, each represented by a [TreeNode](#). To use a tree view, an instance of *TreeView* must first be created using the new operator and (optionally) filled with [TreeNode](#) objects, as illustrated below.

 The *TreeView* class is not supported by Java or thin client applications.

Sample TreeView



Example: Creating TreeView

The following code creates a tree view and fills it with nodes.

```
dcl
*> treeview objects <*
TRVW      object type TreeView;
TRND      object type TreeNode;
TRND2     object type TreeNode;
enddcl
*> create TreeView <*
map new TreeView('TRVW_EXAMPLE') to TRVW
*> create TreeNode and add it to TreeView <*
// Create a root node and attach it to the treeview
map new TreeNode('ROOTNODE1') to TRND
TRND.SetText('Root1')
TRVW.Add(TRND)
// Create a root sibling node and attach it to the treeview
map new TreeNode('ROOTNODE2') to TRND2
TRND2.SetText('Root2')
TRVW.Add(TRND2)
```

Properties and Methods



Inherits [GuiObject](#) and exposes all its properties and methods (not listed below).

The following table lists own properties and methods for the *TreeView* object:

TreeView object properties and methods

Property:Type	Get Method	Set Method
LabelEdit:Boolean	getLabelEdit:Boolean	setLabelEdit(Boolean)
ImageIndex:Integer	getImageIndex:Integer	setImageIndex(Integer)
PopupMenu:PopupMenu	getPopupMenu: PopupMenu	setPopupMenu(PopupMenu)
SelectedImageIndex:Integer	getSelectedImageIndex:Integer	setSelectedImageIndex(Integer)
SelectedNode: TreeNode	getSelectedNode: TreeNode	setSelectedNode(TreeNode)
Text:String	getText:String	setText(String)

LabelEdit:Boolean

The property gets or sets a value indicating whether the label text of the tree nodes can be edited.

ImageIndex:Integer

The property gets or sets the image-list index value of the default image that is displayed by the tree nodes.

SelectedImageIndex:Integer

The property gets or sets the image list index value of the image that is displayed when a tree node is selected.

SelectedNode:[TreeNode](#)

The property gets or sets the tree node that is currently selected in the tree view control.

Additional Action methods

Additional Action Method

Add([TreeNode](#))

AddBmpToImageList(String)
Clear()
Collapse()
CollapseAll()
Count():Integer
Expand()
ExpandAll()
Find(String): TreeNode
Insert(Integer, TreeNode)
Remove(String)

Add([TreeNode](#))

The method adds tree node to the object.

Clear()

The method clears the list of tree nodes of the object.

Collapse()

The method collapses selected tree node of the object.

CollapseAll()

The method recursively collapses all tree nodes of the object.

Count():Integer

The method returns count of tree nodes of the object.

Expand()

The method expands selected tree node of the object.

ExpandAll()

The method recursively expands all tree nodes of the object.

Find(String):[TreeNode](#)

The method finds tree node of the object by its HpsID.

Insert(Integer, [TreeNode](#))

The method inserts tree node to the given position in the list of tree nodes of the object.

Remove(String)

The method finds tree node of the object by its HpsID and removes it.

Events

The *TreeView* object triggers the following events:

- NodeClick
- NodeDoubleClick
- BeforeLabelEdit
- AfterLabelEdit
- BeforeNodeCollapse
- AfterNodeCollapse
- BeforeNodeExpand

- AfterNodeExpand
- BeforeNodeSelect
- AfterNodeSelect

TreeNode

The *TreeNode* represents a node of a [TreeView](#). To use a tree node, an instance of *TreeNode* must first be created using the new operator and added to [TreeView](#) object, as illustrated in the [TreeView code example](#) (also see [TreeView look example](#)).

Both *TreeNode* and [TreeView](#) classes are containers of tree nodes. Thus, tree nodes can be added to *TreeNode* object in the same manner as they are added to [TreeView](#) object (see [TreeView code example](#)).



The *TreeNode* class is not supported by Java or thin client applications.

Properties and Methods

The following table lists own properties and methods for the *TreeNode* object:

TreeNode object properties and methods

Property:Type	Get Method	Set Method
Background:Color		setBackground(Color)
Font:Font		setFont(Font)
Foreground:Color		setForeground(Color)
HpsID:String		setHpsID(String)
ImageIndex:Integer	getImageIndex:Integer	setImageIndex(Integer)
PopupMenu:PopupMenu	getPopupMenu: PopupMenu	setPopupMenu(PopupMenu)
SelectedImageIndex:Integer	getSelectedImageIndex:Integer	setSelectedImageIndex(Integer)
Text:String	getText:String	setText(String)

ImageIndex:Integer

The property gets or sets the image list index value of the image displayed when the tree node is in the unselected state.

SelectedImageIndex:Integer

The property gets or sets the image list index value of the image that is displayed when the tree node is in the selected state.

Additional Action methods

Add(TreeNode)
Clear()
Collapse()
CollapseAll()
Count():Integer
Expand()
ExpandAll()
Find(String): TreeNode
Insert(Integer, TreeNode)
Remove(String)

Add([TreeNode](#))

The method adds tree node to the object.

Clear()

The method clears the list of tree nodes of the object.

Collapse()

The method collapses selected tree node of the object.

CollapseAll()

The method recursively collapses all tree nodes of the object.

Count():Integer

The method returns count of tree nodes of the object.

Expand()

The method expands selected tree node of the object.

ExpandAll()

The method recursively expands all tree nodes of the object.

Find(String):TreeNode

The method finds tree node of the object by its HpsID.

Insert(Integer, TreeNode)

The method inserts tree node to the given position in the list of tree nodes of the object.

Remove(String)

The method finds tree node of the object by its HpsID and removes it.

Events

The *TreeNode* object does not trigger any event.

Supporting Objects

Other objects that support the user interface objects in ObjectSpeak are:

- [Accelerator](#)
- [Color](#)
- [Constants](#)
- [Dimension](#)
- [Font](#)
- [Formats \(Derived\)](#)
- [GlobalEvent](#)
- [Locale](#)
- [Point](#)
- [SetItem](#)
- [Set](#)

Accelerator

The *Accelerator* object is used to allow key combinations that can be assigned to menu items, so that when the key combination is pressed, the menu item is triggered.



Accelerator object is *not* supported in thin client.

There are no predefined *Accelerator* objects: an *Accelerator* object must be created explicitly in order to use it. To create it, specify a character (such as N) and one or more modifiers (SHIFT, CTRL, ALT). Alternatively, specify a virtual key (such as VK_F4, which represents the F4 key) and one or more modifiers. If multiple modifiers are used, they must be numerically added together.

Constants and Methods

The following table lists the constants and methods for this object:

Accelerator constants and methods

Modifier	Description
SHIFT	Shift key
CTRL	Ctrl key
ALT	Alt key
Virtual Key	Description
VK_F1	F1 key
VK_F2	F2 key
VK_F3	F3 key
VK_F4	F4 key
VK_F5	F5 key
VK_F6	F6 key
VK_F7	F7 key
VK_F8	F8 key
VK_F9	F9 key
VK_F10	F10 key
VK_F11	F11 key
VK_F12	F12 key

accelerator(Char :Character; Modifiers:Integer)	Constructor method, refer to the example for usage.
accelerator(VirtualKeyCode:Integer; Modifiers:Integer)	Constructor method, refer to the example for usage.
KeyCode:Integer	(read only)
KeyChar:Integer	(read only)
Modifiers:Integer	(read only)

Example: Assigning an Accelerator

When using these constants, you must indicate that they are defined in either the [Accelerator](#) class or the [Constants](#) class. To assign an accelerator of Ctrl+N to a menu item that creates new files:

```
NewMenuItem.SetAccelerator(  
new Accelerator('N', Accelerator.CTRL))
```

To specify an accelerator of Ctrl+Alt+F1, you could do the following:

```
NewMenuItem.SetAccelerator(  
new Accelerator(Accelerator.VK_F1,  
Accelerator.CTRL + Accelerator.ALT))
```

Notice that the CTRL and ALT modifiers are added together in order to specify that both Ctrl and Alt must be pressed with the F1 key. The above example also illustrates that the accelerator constants are defined in the Constants class as well as the Accelerator class.

Color

The *Color* object is used to specify the foreground and background colors of objects.

Constructor and Parameters

You can create new *Color* objects with specified RGB (red, green, and blue) components, as shown in the following code sample:

```
NameField.SetForeground(new Color(128,128,255))
color(Red:Integer; Green:Integer; Blue:Integer)
```

A *Color* object can be created and assigned to a local variable, so you can use it multiple times:

```
dcl
CustomColor object type Color;
enddcl
map new Color(128,128,255) to CustomColor
NameField.SetForeground(CustomColor)
AddressField.SetForeground(CustomColor)
```

See [Constructor and Parameters](#) for Font for a discussion of dynamic font creation.

When using predefined colors, indicate that they are defined in the *Color* class, as shown in this example:

NameField.SetForeground(Color.RED)

Constants and Methods

The following table lists the constants and methods for this object:

Color object constants and methods

BLACK	WHITE
DARKBLUE	BLUE
DARKGREEN	GREEN
DARKCYAN	CYAN
DARKRED	RED
DARKMAGENTA	MAGENTA
DARKYELLOW	YELLOW
DARKGRAY	GRAY
LIGHTGRAY	PINK
BROWN	TURQUOISE

[RGB:Integer](#)

[Color\(Integer, Integer, Integer\)](#)

[getBlue\(\):Integer](#)

[getGreen\(\):Integer](#)

[getRed\(\):Integer](#)

RGB:Integer

This allows you to get the RGB() value of the color. The RGB() value can be used to create a Java color or to compare two color objects. Here is an example of these uses:

```
dcl
brownColor object type 'java.awt.Color';
enddcl
map new 'java.awt.Color'(Color.BROWN.RGB()) to brownColor
if brownColor.RGB() = Color.BROWN.RGB()
trace('color matches');
endif
```

Color(Integer, Integer, Integer)

This method is a constructor taking red, green, and blue as integers.

getBlue():Integer

This method gets the blue value.

getGreen():Integer

This method gets the green value.

getRed():Integer

This method gets the red value.

Constants

The *Constants* object defines useful integer constants that can be used within the rule.

Constants

The following table provides a list of the constants.

Constants methods

ACCEPT	OK_CANCEL
ALL_FIRST_UPPER_CASE	OUT_OF_RANGE
ALT	PLAIN
ASYNC_EVENT*	PLAIN_MENUITEM
BORDER_DIALOG	QUESTION
BORDER_NONE	ROLLBACK
BORDER_SIZEABLE	SETDOMAIN
BOTTOM	SHIFT
CANCEL	SHOW_ALWAYS
CENTER	SHOW_AS_NEEDED
COORDINATE_CHAR	SHOW_NEVER
CHECKBOX_MENUITEM	SINGLE_RANGE_SELECTION
COORDINATE_PIXEL	SINGLE_SELECTION
CTRL	SYSTEM_EVENT*
DEFAULT_BUTTONS	TOP
DEFAULT_CASE	UP
DOWN	UPPER_CASE

ERROR	USER_EVENT*
FIRST_UPPER_CASE	VERTICAL_LINES
HORIZONTAL_AND_VERTICAL_LINES	VIEWDOMAIN
HORIZONTAL_LINES	VK_F1
INFORMATION	VK_F10
INTERFACE_EVENT*	VK_F11
IN_ERROR	VK_F12
LANDP_EVENT*	VK_F2
LANDP_REQUEST_EVENT*	VK_F3
LANDP_SYSTEM_EVENT*	VK_F4
LEFT	VK_F5
LISTBOX_BOTTOM	VK_F6
LISTBOX_TOP	VK_F7
LOWER_CASE	VK_F8
MULTIPLE_RANGE_SELECTION	VK_F9
NO	WAIT
NOWAIT	WARNING
NO_LINES	YES
OK	YES_NO
OK_BUTTON	YES_NO_CANCEL
RIGHT	

Entries marked with an asterisk are small Integer type constants. All other constants are Integer type.

GuiObject Type Constants

GuiObject type constants

BITMAP	MENU
CHECKBOX	MENUITEM
COLUMN	MULTILINEEDIT
COMBOBOX	PASSWORDFIELD
EDITFIELD	POPUPMENU
ELLIPSE	PUSHBUTTON
FILEEDITOR	RADIOBUTTON
GROUPBOX	RECTANGLE
LABEL	TABLE
LISTBOX	WINDOW

Format Type Constants

- DATE_FORMAT
- DECIMAL_FORMAT
- LONGINT_FORMAT
- SHORTINT_FORMAT
- STRING_FORMAT

- TIME_FORMAT

Usage

To map the value to a field in a view and pass it to other rules, create a Field of data type Smallint or Integer. For example:

```
dcl
SaveMessageBox object type MessageBox;
MessageBoxReturn Integer;
enddcl
> create a message box <
map new MessageBox to SaveMessageBox
> set message box properties <
SaveMessageBox.SetMessageType(Constants.QUESTION)
SaveMessageBox.SetButtonType(Constants.YES_NO_CANCEL)
SaveMessageBox.SetTitle('Save File')
SaveMessageBox.SetMessage('Save the file named %1?')
SaveMessageBox.SetArgument1('SAMPLE.XML')
> display message box <
map SaveMessageBox.Show to MessageBoxReturn
> map the value back to the calling rule <
map MessageBoxReturn to smallint_field of rule_output_view
```

Another example:

```
map Constants.LEFT to <edit_field_hpsid>.justification
```

Dimension

The *Dimension* object is used to specify the height and width of any visible GUI object (in integer precision) including a windows. In particular, the [Size:Dimension](#) property of a visible object is of type Dimension.

Normally the values of width and height are non-negative integers. The constructor that allows you to create a *Dimension* does not prevent you from setting a negative value for these properties. If the value of width or height is negative, the behavior of some methods defined by other objects is undefined.



Dimension object is *not* supported in thin client.

Constructor and Parameters

The *Dimension* object constructs a Dimension and initializes it to the specified width and height.

Dimension(int width, int height)

Properties and Methods

The following table lists the properties and methods for this object.

Dimension properties and methods

Property:Type (Get Method)	Set Method
Height:Integer	setHeight(Integer)
Width:Integer	setWidth(Integer)

Height:Integer

This is the height component of a [Dimension](#) object. Dimension objects are used to specify, via the [Size:Dimension](#) property, the height and width of all visible objects, including the window itself. Use this property to query or set the height component of the [Dimension](#).

Width:Integer

This property specifies the width of a [Dimension](#) object. The width of the column is set and queried with this property. Use this property to query or

set the width component of the [Dimension](#).

Example: Resizing an Edit Field

To resize an edit field when a push button is pressed, use the following code:

```
proc ResizeButtonClick for Click object ResizeButton
(e object type ClickEvent)
NameField.SetSize(new Dimension(300, 100))
endproc
```

To set the dimensions of an object:

```
dcl
myEditDimension object type Dimension;
enddcl
proc InitProc for Initialize object TEST_DIMENSION (e object type InitializeEvent)
map new dimension(25,40) to myEditDimension EDIT_HPSID.setSize(myEditDimension)
endproc
```

Font

The *Font* object specifies the font used to display text.



Font object is *not* supported for thin client.

Constructor and Parameters

You can create new *Font* objects with specified font names, styles, and sizes using the following code sample:

NameField.SetFont(new Font('Arial' , Font.BOLD, 14))

A *Font* object can be assigned to a local variable to use multiple times.

```
dcl
GroupBoxFont object type Font;
enddcl
map new Font('Arial' , Font.BOLD, 14) to GroupBoxFont
myGroupBox.setFont(GroupBoxFont)
```

When using predefined fonts, indicate that they are defined in the *Font* class.
Font styles can be: FONT.PLAIN, FONT.BOLD, FONT.ITALIC, or FONT.BOLD+FONT.ITALIC.

Constants and Methods

Font constants and methods

The following table lists the constants and methods for this object:

BOLD	ROMAN18
ITALIC	ROMAN24
MODERN8	SWISS8
MODERN10	SWISS10
MODERN12	SWISS12
PLAIN	SWISS14
ROMAN8	SWISS18

ROMAN10	SWISS24
ROMAN12	SYSTEMFONT8
ROMAN14	

Font properties and methods

The following table lists the properties and methods for this object:

Property:Type (Get Method)	
displayName :String	getFont(FontName:String):Font getStyle():Integer getSize():Integer

The displayName is the logical name of the font (the name parameter used to construct the font) or the name set by the user using the property.

```
map "my font" to <afont>.displayname
```

getFont(FontName:String):Font

This method returns a predefined font (fonts defined in the fonts.ini). For example, if *TIMES* is defined in font.ini as:

```
[TIMES]
Java=sanserif,12
```

then the following statement returns sanserif, 12 point for *myFont* :

```
set myFont := Font.getFont("TIMES")
```

getStyle():Integer

Returns the current style. This is a combination of Font.PLAIN, Font.BOLD, Font.ITALIC.

getSize():Integer

Returns the size of the font.

Example: Specifying the font class

Indicate the font class when using predefined fonts, as shown here:

```
NameField.SetFont(Font.SWISS14)
```

Formats (Derived)

These derived formats inherit all the properties from the [Format](#) object.

- DecimalFormat
- LongIntFormat
- IntFormat
- ShortIntFormat
- FloatFormat
- DoubleFormat
- DateFormat
- TimeFormat
- TimestampFormat
- StringFormat

LongIntFormat class is used to format LongInt data type (64-bit integer), IntFormat class is used to format Int Format data (32-bit integer), and ShortIntFormat class is used to format ShortInt data(16-bit integer). The other format classes are FloatFormat, DoubleFormat, DecimalFormat, DateFormat, TimeFormat, TimestampFormat and StringFormat, which are associated with decimal, date, time, and string data types.

For a list of valid formatting symbols, refer to the *Rules Language Reference Guide* .

Properties and Methods

The following table lists the properties and methods for the derived format objects:

Derived format properties and methods

Property:Type (Get Method)	Set Method
These are available for numeric types: DecimalFormat, LongIntFormat, IntFormat, ShortIntFormat, DoubleFormat, FloatFormat	
	setMinimum(ShortInteger)
	setMinimum(Integer)
	setMinimum(Decimal)
	setMinimum(Double)
	setMaximum(ShortInteger)
	setMaximum(Integer)
	setMaximum(Decimal)
	setMaximum(Double)
Currency:Boolean	setCurrency(Boolean)
This is available for string types: StringFormat	
Case	setCase(Integer)

The setMinimum() and setMaximum methods set the range for a numeric field. For example, the value range of a field can be set to a minimum of 0 and a maximum of 100, and values less than 0 or greater than 100 will result in an error.

The setCase() method for a string sets the case of the characters in the string (uppercase or lowercase). The following constants can be used with the setCase method:

- *UPPER_CASE* - All characters are uppercase
- *LOWER_CASE* - All characters are lowercase
- *ALL_FIRST_UPPER_CASE* - The first letter of every word is uppercase
- *FIRST_UPPER_CASE* - Only the first letter of the first word is uppercase
- *DEFAULT_CASE* - Use the case as entered and this is the default

For example:

```
dcl
strFormat object type StringFormat;
enddcl
map new StringFormat to strFormat
strFormat.setCase(Constants.FIRST_UPPER_CASE)
```

GlobalEvent

The *GlobalEvent* object is used for posting an event to which applications can subscribe. Global eventing is supported in Java (thick) clients only, but not in C# clients.

Properties and Methods

The following property is supported for this object:

```
post(Rule InstanceName:String):Boolean
```

For example, an ObjectSpeak method post could be used on the GlobalEvent object using

```
THRESHHOLD_MET.post ( )
```

where THRESHHOLD_MET is the name of the Physical Event object under a Rule.

On the subscribing end, the application can use either an event procedure or a Converse Event to handle the event. If the pre-defined system view, HPS_EVENT_VIEW, is attached to the subscribing Rule, an event listener is automatically added to the Rule and a ConverseEvent is triggered on receiving this event. The EVENT_NAME of the HPS_EVENT_VIEW has the name of the event.

Locale

The *Locale* object is used to specify the information about country and language. Also, see the [Locale:Locale](#) property.

Constructor and Parameters

The default constructor creates a System locale object (the same as HpsLocale.SYSTEM). The method name is *Constructor*.

Locale()

The user-configurable constructor allows you to specify the language and country code and creates a Locale object. This constructor creates a Locale from the parameters for Language, a two-digit ISO language code (en for English, for example), and Country, a two-digit ISO country code (US for United States, for example).

Locale(aLanguage:String, aCountry:String)

Properties and Methods

The following *Locale* objects are predefined:

Locale pre-defined objects

ALBANIA	LUXEMBOURG
ARGENTINA	NETHERLAND
AUSTRALIA	NEW_ZEALAND
AUSTRIA	NORWAY
BELGIUM	POLAND
BRAZIL	PORTUGAL
CANADA_FRENCH	ROMANIA
CANADA_ENGLISH	SINGAPORE
CHINA	SOUTH_AFRICA
CZECHOSLOVAKIA	SOUTH_KOREA
DENMARK	SPAIN
FINLAND	SWEDEN
FRANCE	SWITZERLAND
GERMANY	TAIWAN
GREECE	THAILAND
HONGKONG	THAILAND_BUDDHIST
HUNGARY	TURKEY
ICELAND	UNITED_KINGDOM
IRELAND	UNITED_STATES
ITALY	YUGOSLAVIA
JAPAN	SYSTEM

The following table lists the read-only properties for this object:

Locale properties and methods

Property:Type (Get Method)	Description
Country :String	Gets country name
CurrencySymbol :String	Gets symbol used for currency
DateSeparator :Char	Gets separator (delimiter) for date format display
DecimalSeparator :Char	Gets separator (delimiter) for decimal point
DefaultDateFormat :String	Gets default date display format
DefaultTimeFormat :String	Gets default time display format
Language :String	Gets language for display
ThousandsSeparator :Char	Gets separator for thousands
TimeSeparator :Char	Gets separator (delimiter) for time format display

Point

The *Point* object is used to specify the location (in integer precision) of any visible GUI object, including a window, in (x, y) coordinate space. In particular, the [Location:Point](#) property of visible objects is of type Point. For a window, the X and Y coordinates are relative to the upper left corner of the screen. For user interface objects (such as edit fields) the coordinates are relative to the upper left corner of the part of the window below the title and menu bar.



Point object is *not* supported in thin client.

Constructor and Parameters

This object constructs and initializes a point at the specified (x, y) location in the coordinate space.

Point(int x, int y)

Properties and Methods

Here are the properties and methods for this object:

Point properties and methods

Property:Type (Get Method)	Set Method
X:Integer	setX(Integer)
Y:Integer	setY(Integer)

X:Integer

This is the horizontal position (x coordinate) property of a Point object. Point objects are used in the [Location:Point](#) property to specify the location of any object, including the window itself. The X property is used to query or set the x-component of the location.

Y:Integer

This is the vertical position (y coordinate) property of a Point object. Point objects are used in the [Location:Point](#) property to specify the location of any object, including the window itself. The Y property is used to query or set the y-component of the location.

Example: Repositioning an Edit Field

Use the following code to reposition an edit field when a push button is pressed:

```

proc MoveButtonClick for Click object MoveButton
(e object type ClickEvent)
NameField.SetLocation(new Point(200, 200))
endproc

```

The following example shows how to edit the location:

```

dcl
myEditLocation object type Point;
enddcl
proc InitProc for Initialize object TEST_DIMENSION (e object type InitializeEvent)
map new Point(125,40) to myEditLocation
EDIT_HPSID.setLocation(myEditLocation)
endproc

```

SetItem

A SetItem object is used to dynamically create or update Rules Language SET elements, also known as set items. See also [Set](#).

Constructors and Parameters

The following statements create SetItem objects:

SetItem(display: String, encoding: DataObject)

Specifies a new set item display.

SetItem(display: String, encoding: DataObject, state: Integer)

Specifies a new set item state.

SetItem(display: String, encoding: DataObject, text: String, state: Integer)

Specifies new set item text.

By including different parameters, you can create new SetItem objects in three ways. These parameters are:

- The display parameter, which specifies a new set item display
- The encoding parameter, which specifies new set item encoding
- The state parameter, which specifies a new set item state. The default value for this parameter is ENABLED.
- The text parameter specifies new set item text. The default value for this parameter is an empty (null) string.

Examples of use:

```

dcl
newItem OBJECT TYPE SetItem;
enddcl
MAP NEW SetItem("display", "encoding") TO newItem
MAP NEW SetItem("display", "encoding", SetItem.DISABLED) TO newItem
MAP NEW SetItem("display", "encoding", "text", SetItem.ENABLED) TO newItem

```

Properties and Constants

state:Integer

This property reflects the set item state, which is the way an item appears in associated GUI control. Set item can be in one of three states:

- Enabled (default)
- Non-selectable. The item does not display in the drop-down list but displays if the encoding was entered programmatically. This is a type of read-only set item.
- Disabled. The item cannot be selected and is not displayed. It is as though the item does not exist in the set.

The three corresponding constants in SetItem are:

- ENABLED
- NONSELECTABLE

- DISABLED

Example of use:

```
newItem.setState(SetItem.DISABLED)
```

Methods

getEncoding(): DataObject

Returns encoding of the set item.

getDisplay(): String

Returns display of the set item.

getText(): String

Returns text of the set item.

Set

A *Set* object is used to update Rules Language SETs dynamically. Use this object to add new elements, access set items at runtime, and to read and change their attributes. New sets cannot be created at runtime. A Set object is constructed automatically for each set entity attached to the rule and has a the same name as the set entity long name.

Properties

Count:Integer

Returns the number of items in the set.

For example:

```
DCL
  SI OBJECT TYPE SetItem;
  I,N INTEGER;
ENDDCL
DO FROM 1 TO mySet.Count INDEX I
  MAP mySet.GetSetItemAt(I) TO SI
  TRACE("INDEX= ",I," ITEM DISPLAY: ",SI.GetDisplay())
ENDDO
```

Methods

addSetItem(item: SetItem)

Adds specified and previously created set item to the set. Encoding of item must be of the same type as the set or of convertible type (see [Convertible types](#)). No checking for duplicates is performed.

For example:

```
mySet.addSetItem(new SetItem(display, encoding))
```

Or to add the same set item to several sets, write:

```
MAP NEW SetItem(display, encoding) TO newItem
mySet1.addSetItem(newItem)
mySet2.addSetItem(newItem)
```

addSetItem(display: String, encoding: DataObject)

Adds a new set item to a set.

addSetItem(display: String, encoding: DataObject, state: Integer)

Adds a new set item with a state parameter specified.

addSetItem(display: String, encoding: DataObject, text: String, state: Integer)

Adds a new set item with a text parameter specified.

These methods add a new set item to the set. The display parameter specifies new set item display. The encoding parameter specifies new set item encoding. The state parameter specifies new set item state. The text parameter specifies new set item text. If addSetItem is used without a state parameter specified, it defaults to SetItem.ENABLED. If addSetItem without text parameter is used, it defaults to empty string.

Encoding given must be of the same type as the set or of convertible type (see [Convertible types](#)). If type of given encoding is not completely equal to set type (but is convertible), this encoding is converted to set type. No checking for duplicates is performed.

getSetItem(encoding: DataObject) : SetItem

This method searches within the set for a set item with specified encoding. If several set items exist with the same encoding, the first one found is returned. SetItem object is returned if a set item with this encoding is found; otherwise, a null reference is returned. Use the isClear function to test if the method returns null reference.

getSetItemFromDisplay(display: String) : SetItem

This method searches within the set for a set item with a specified display. If several set items exist with equal displays, the first one found one is returned. SetItem object is returned if a set item with this display is found; otherwise, a null reference is returned. Use the isClear function to test if the method returns null reference..

getSetItemAt(index:Integer) : SetItem

This method returns the set item with the specified index. Null reference is returned if the index is greater than the number of set items. Use the isClear function to test if the method returns null reference.

refresh()

This method updates GUI control associated with the set. Use this method after adding new items or changing a status of one or several set items.

Example

Assume a rule has attached set of type INTEGER with name account_type.

```
DCL
display, encoding VARCHAR(40);
text VARCHAR(2000);
newSet (100) VIEW CONTAINS display, encoding, text;
setSize, i INTEGER;
item OBJECT TYPE SetItem;
ENDDCL
SQL ASIS
//query a set from database, populate newSet view and setSize
ENDSQL
DO TO setSize INDEX I
account_type.addSetItem(newSet.display,
INT(newSet.encoding)
)
// OR
// account_type.addSetItem(NEW SetItem(newSet.display,
// INT(newSet.encoding))
ENDDO
account_type.refresh()
```

Convertible types

Source Type	Target Type	Converted Type	Comments
INTEGER	DEC(n, m)	DEC(10, 0)	No data loss if n >= 10.
INTEGER	SMALLINT	SMALLINT	Possible data loss.
SMALLINT	DEC(n, m)	DEC(5, 0)	No data loss if n >= 10.

SMALLINT	INTEGER	INTEGER	No data loss.
DEC	SMALLINT	SMALLINT	Possible data loss.
DEC	INTEGER	INTEGER	Possible data loss.
Any character type	Any character type	Equal to target type	Possible data loss if target character data type is shorter than source.

Java Batch Objects

Batch objects are used to support batch client applications on the Java platform. This section discusses the `Rule` object and events for ObjectSpeak batch objects.

The only class of objects in batch ObjectSpeak is the [Abstract Class Objects](#).



For information on how to create these objects using the Window Painter, refer to the Window Painter tool topic in the *Development Tools Reference Guide*.

Abstract Class Objects

The following objects are the Abstract Class (high-level) batch objects available in ObjectSpeak:

- [Rule](#)
- [System](#)

Rule

The `Rule` object plays a central role in the AppBuilder Java batch because it provides an object interface to AppBuilder Rules Language rules. The `Rule` object has no properties, but it does define a number of methods to initiate actions, obtain information, and implement events.

This section includes:

- [Rule Methods](#)
- [Events](#)

Rule Methods

The following table lists the methods for the `Rule` object.

Rule object methods

<code>queryUserAuthentication():Boolean</code>
<code>setUserAuthentication(userID:String, password:String)</code>
<code>trace(message:String<, object:view OR object:field>)</code>

`queryUserAuthentication():Boolean`

The `queryAuthentication():Boolean` method queries for the name of the user and verifies permissions.

`setUserAuthentication(userID:String, password:String)`

The `setUserAuthentication()` method enables you to set the user credentials to authenticate the remote server rules. The same method is invoked if `QUERY_AUTHENTICATION_ON_STARTUP` is enabled through the setting in the `APPBUILDER.INI` file. The AppBuilder communication exits can override this information when a remote rule is invoked.

`trace(message:String<, object:view OR object:field>)`

The `trace(message:String)` method traces the message. This method accepts either a `View` or a `Field` as an optional second parameter. When the second parameter is specified, the name and the value of the specified object is appended to the trace message. If a `View` is specified as the second parameter, the name and the value of all fields are added as separate lines in the trace output.

Events

The following events can be triggered using the `Rule` object:

- `CommError` - not supported in C#
- `SQLException` - not supported in C#

System

The `System` object provides an interface for some system services. Two methods are used to translate long names of AppBuilder entities, such as rules or views, into corresponding Java class names or Java object names according to the AppBuilder naming conventions.

It is not supported in C#.

Constants and Methods

Constants

Type values:

The following table shows the various system type values.

System type values

<code>RULE_TYPE</code>	<code>COMPONENT_TYPE</code>
<code>VIEW_TYPE</code>	<code>WINDOW_TYPE</code>
<code>VIEWARRAY_TYPE</code>	<code>OBJECT_TYPE</code>
<code>SET_TYPE</code>	<code>HPSID_TYPE</code>
<code>FIELD_TYPE</code>	

Methods:

```
longNameToClassName(type:Integer, longName:String) : String
```

```
longNameToObjectName(type:Integer, longName:String) : String
```

```
longNameToClassName(type:Integer, longName:String) : String
```

The `longNameToClassName` method translates the long name of an AppBuilder entity, whose type is specified by the first parameter using one of the `System` object constants, into a corresponding Java class name according to AppBuilder naming conventions.

For example:

```
SET RuleClassName := System.longNameToClassName(System.RULE_TYPE, "MY_RULE")
RuleCaller.ExecuteRule(RuleClassName)
```

The `type` parameter can have only one of the following values: `RULE_TYPE`, `VIEW_TYPE`, `VIEWARRAY_TYPE`, `SET_TYPE` and `COMPONENT_TYPE`; other entities do not generate a class.

```
longNameToObjectName(type:Integer, longName:String) : String
```

The `longNameToObjectName` method translates the long name of an AppBuilder entity, whose type is specified by the first parameter using one of the `System` object constants, into a corresponding Java object name according to the AppBuilder naming conventions.

For example:

```
SET RuleClassName := System.longNameToClassName(System.RULE_TYPE, "MY_RULE")
SET ViewObjectName := System.longNameToObjectName(System.VIEW_TYPE, "MY_VIEW")
```


RuleClassName ++ ViewObjectName forms a reference to a field of generated rule class, which corresponds to the instance of view MY_VIEW owned by MY_RULE.

The type parameter can be anything except COMPONENT_TYPE. This is because components' classes are never instantiated, and there is no corresponding property in the rule class. Use HPSID_TYPE for windows' objects that have HPSID, but OBJECT_TYPE for other objects and aliases.

Events

Events

This topic gives detailed information on all the events that are generated for Java-based ObjectSpeak and describes the event objects that are passed into the event procedures.

Each event procedure has only one parameter. The name of this parameter is the name of the event followed by the word *Event*. For example, the parameter for the Click event is called *ClickEvent*.

Event parameters are objects and, as such, have properties. Many of these properties are read-only ? that is, you can query their value but not assign a value to them. For convenience, non-read-only properties have corresponding set methods.

Events that are not supported for thin clients (HTML) are noted. Migration samples for all the events are provided at < *AppBuilder* >\samples\java\ospk.zip

For more information on the error checking and validating events, refer to [Data Validation](#).

Data Validation

Data Validation

AppBuilder ObjectSpeak enables you to validate data when windows are created in an application. Data validation uses events to notify the application that data is invalid or missing. AppBuilder provides both field-level and window-level validation.

- [Field-level Validation](#) verifies that a field contains syntactically-valid data that is acceptable to the application.
- [Window-level Validation](#) permits a window to be closed only if all the mandatory fields contain data and the data in all fields contains valid syntax.

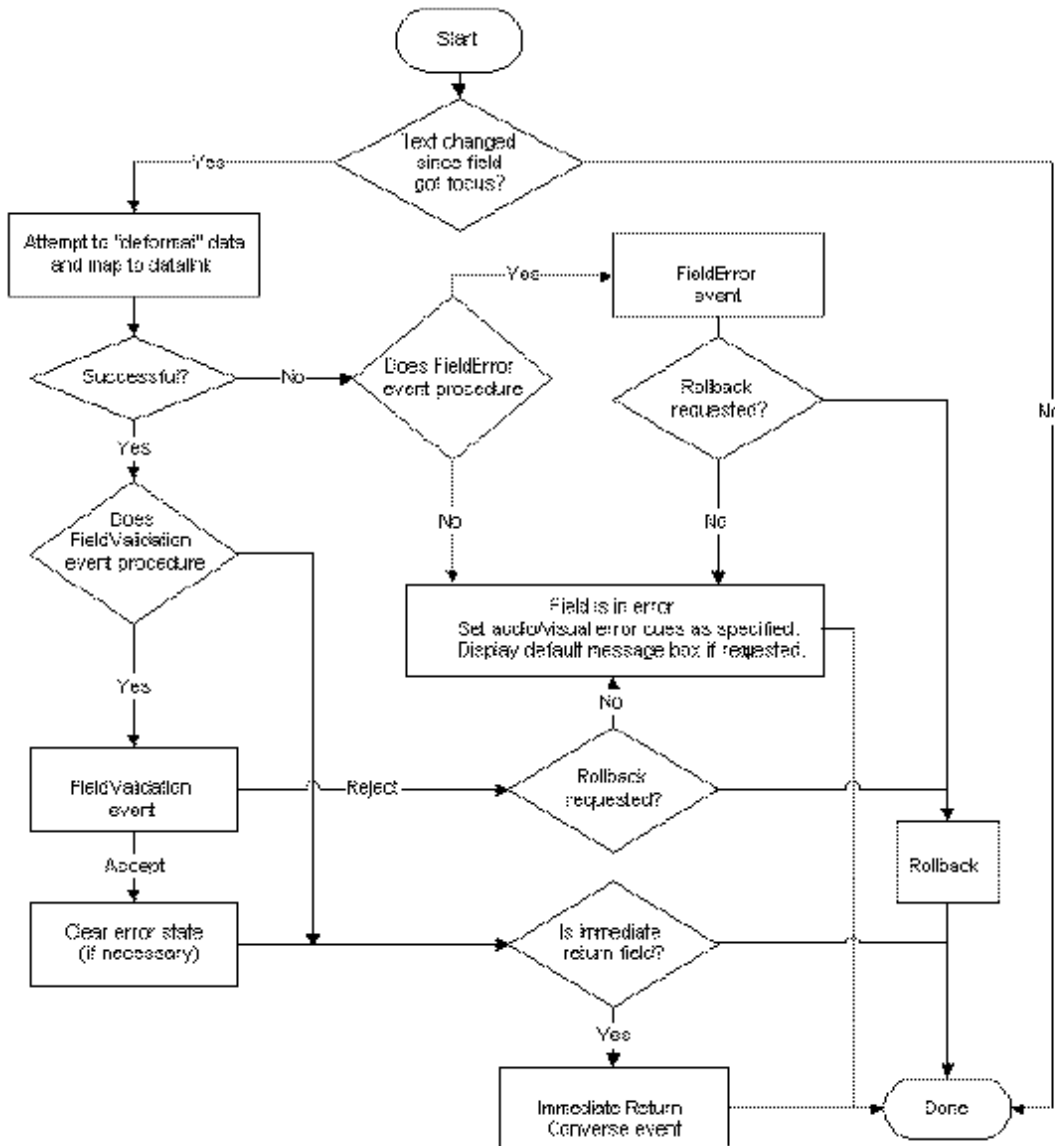
Field-level Validation

Field-level data validation occurs when:

- The data in an editable field changes and focus is shifted away from the field
- The end user presses **Enter** while an editable field has focus

The following figure describes the logic used to validate fields and windows.

Validation Flow Diagram



Field-level validation occurs in two stages:

Triggering the FieldError Event - when data in the field contains syntax errors, the system triggers the **FieldError** event.

1. **Triggering the FieldValidation Event** - if the data is syntactically correct, the system maps the data to the data-linked data object (if applicable) and triggers the **FieldValidation** event.

If a field contains errors, end users can return the field to the last known valid value by pressing Escape (**Esc**) while the field has focus. Field-level data validation occurs for the following objects with editable fields:

- [ComboBox](#)
- [EditField](#)
- [MultiLineEdit](#)
- [PasswordField](#)
- [Table](#)

Triggering the FieldError Event

The **FieldError** event is triggered if the field contains errors, allowing the application to specify:

- If the application should roll back the data to the last known acceptable value or leave the field in error. Use the [Rollback: Boolean](#) property to specify the required action. The default value is *False* and the data is *not* rolled back.
- If the application should display a message box describing the error. Use the [ShowMessage: Boolean](#) property to specify the action. By default, the value is *True* and the system shows the message box.

The **FieldError** event contains the following important properties:

- [HpsID:String](#) Returns the system identifier (HPSID) of the field in error.
- [Source:GuiObject](#) Provides a reference to the field so that its properties can be accessed and its methods called, as necessary.

Example: FieldErrorEvent

The following is a sample *FieldErrorEvent* procedure for an edit field named StartDate, which is data-linked to a Date field in the application hierarchy:

```
proc for FieldError object StartDate

(FldErr object type FieldErrorEvent)
set FldErr.Rollback = TRUE
Trace (' HpsID =', FldErr.HpsId)
Trace (' Source HpsID =', Fldr.Source.HpsId)
Trace (' ShowMessage =', FldErr.ShowMessage)
endproc
```

This procedure is called if the end user enters an invalid date into the StartDate field, and it responds by rolling the data back to the last known acceptable value.

Triggering the FieldValidation Event

The [FieldValidation](#) event is triggered when an editable field loses focus or when the end user presses **Enter** while a field has focus if there are no syntax errors. It allows the application to specify if data should be:

- Accepted
- Rolled back to the last known acceptable value
- Considered in error

For example, a field containing an interest rate may have a value that is syntactically correct. In other words, it contains a valid numeric format, but is invalid because the specified interest rate is not within a pre-defined range.

This event also allows you to specify that the application display a message box describing the error. Use the [ShowMessage:Boolean](#) property to specify the required action. By default, the system shows the message box.



If the data is accepted or rolled back, no message box is shown, regardless of the value of *ShowMessage*.

The *FieldValidation* event contains the following important properties:

- [HpsID:String](#) ? Returns the system identifier (HPSID) of the field in error.
- [Source:GuiObject](#) ? Provides a reference to the field allowing you to access its properties and call its methods, if necessary.

The data is mapped to an existing data link before the FieldValidationEvent is initialized, allowing data to be examined within the event procedure or changed with a map statement.

Example: FieldValidation Event

The following is a sample *FieldValidation* event procedure for an edit field named "Interest" that is data-linked to a Decimal field in the application hierarchy:

```

proc for FieldValidation object Interest

( e object type FieldValidationEvent )
if (MAIN_WINDOW_VIEW.Amount < 10000) and
(MAIN_WINDOW_VIEW.Interest < 10 )
*> display a message box <*>
DisplayMessageBox( 'Higher interest is required' )
\*> indicate that data is not acceptable; field
is now in error <\*>
e.SetResponse( Constants.IN_ERROR )
*> suppress the default message box <*>
e.SetShowMessageBox( False )
endif
endproc

```

When the end user modifies the data in the Interest field and moves the focus to another field, the system calls this event procedure. This procedure examines the loan amount. If the loan amount is less than \$10,000 and the specified interest rate is less than 10%, the event calls a procedure to display a message box indicating that a higher interest rate is required. It then specifies that the data in the field should be considered in error. Depending on the settings in the *APPBUILDER.INI* configuration file, the field displays the error condition by changing the foreground or background color, or both, of the edit field.

Window-level Validation

Window-level validation occurs when the end user clicks a button or selects a menu item whose [Validation:Boolean](#) property is *True* or the [IgnoreValidation:Boolean](#) property is *False*. Window-level validation allows the application to verify that the end user has specified all the required information and that the information in the various fields is acceptable.

[Validation Flow Diagram](#) describes the logic used to validate fields and windows.

Window-level validation occurs in three stages:

Triggering the WindowError Event - The system examines all fields with [Mandatory:Boolean](#) property set to *True* to ensure that they contain data. If any mandatory fields are empty, then the system triggers a [WindowError](#) event. If all mandatory fields contain data, the system proceeds to the second stage.

1. [Triggering the WindowError Event](#) - The system examines the fields to ensure that the data is syntactically correct. If there are syntax errors, the system triggers a [WindowError](#) event.
2. [Triggering the WindowValidation Event](#) - If the data syntax is correct, the system triggers a [WindowValidation](#) event. This allows the application to verify the data.

If window-level validation fails for any reason, then the [Click](#) event of the button or menu item is *not* triggered, thereby preventing the button or menu item from triggering events.

By using the [Validation:Boolean](#) property, you can routinely check mandatory fields as part of window-level validation.

For backwards compatibility, push buttons and menu items in the AppBuilder Java client contain not only the [Validation:Boolean](#) property, but also two Boolean properties: [IgnoreValidation:Boolean](#) and [CheckMandatoryFields:Boolean](#). These validation properties perform slightly different tasks:

- [Validation:Boolean](#) - always checks mandatory fields.
- [IgnoreValidation:Boolean](#) - only checks mandatory fields if the [CheckMandatoryFields:Boolean](#) is *True*.

Triggering the WindowError Event

The system triggers the [WindowError](#) event when:

- Any mandatory fields are empty (if [MandatoryError\(\):Boolean](#) is *True*)
- Any fields contain data errors (if [FieldError\(\):Boolean](#) is *True*)

This event also contains a [ShowMessage:Boolean](#) property that specifies if a message box describing the error displays. By default, the system shows the message box.

Triggering the WindowValidation Event

The system triggers the [WindowValidation](#) event when:

- All mandatory fields contain data and no fields contain errors.

This event defines the following results:

- The required action if the application accepts the data. Use the [Accept\(\):Boolean](#) property to specify the resulting action. If the application accepts the data, the push button or menu option triggers a [Click](#) event.

- That a message box display describing data errors. Use the [ShowMessage:Boolean](#) property to specify the resulting action. By default, the system shows the message box.



If the data is accepted, no message box is shown, regardless of the value of **ShowMessage** .

User-Interface Properties

[User-Interface Properties](#) implemented by the Java user interface objects discussed in previous sections are described in this topic.

ObjectSpeak Events

Events in ObjectSpeak include:

ObjectSpeak events

Activate	FieldValidation	SQLException
CellFocusGained	FocusGained	Terminate (for Rule)
CellFocusLost	FocusLost	Terminate (for Window)
ChildRuleEnd	HeaderClick	Timer
Click	Initialize (for Rule)	WindowError
Close	Initialize (for Window)	WindowValidation
CommError	MessageBox	
Converse	PageSelect	
DataRequired	ParentRuleEnd	
DoubleClick	Post	
FieldError	RuleEnd	

Activate

This event is triggered on an existing instance of a detached rule if another attempt is made to detach that rule with the same instance name. This event has no methods or properties.

Example: Activate Event

The following is an example of the syntax:

```
*> example of Window ActivateEvent <*
proc for Activate object APPB_SS_MCLB_DIS
(e object type ActivateEvent)
endproc
```

CellFocusGained

This event is triggered when a table cell gains focus. Various properties on the event can be used to obtain information about the cell that gained focus.

Properties

The following table describes event properties for CellFocusGained.

CellFocusGained event properties

Property and Type	Description
-------------------	-------------

Column:Column	This provides an object reference to the Column object that contains the cell clicked on. Properties and methods of the Column object can be called to obtain additional information or to perform operations.
ColumnIndex:Integer	This is the order number of the column, where the leftmost column is 1, the one to its right is 2, and so on. The numbering column (if present) is not included in the numbering. Thus if there is a numbering column, the column to its immediate right is 1.
HpsID:String	This returns the system identifier (HPSID) of the object that generated the event.
PhysicalIndex:Integer	This indicates the index (or occurrence number) of the row that contains the cell in the occurring view to which the table is data-linked.
Source:GuiObject	This is a reference to the Table object that generated the event, typed as a GuiObject.
VirtualIndex:Integer	This indicates the virtual row number for the row that contains the cell.

Support

This event is *not* supported for thin (HTML) clients. It is only supported for Java (thick) clients.

Example: CellFocusGained Event

The following is an example of the syntax:

```
Proc for CellFocusGained object TestMclb (MclbFocusGained object type CellFocusGainedEvent)
Trace ('Column HpsID = ' ,MclbFocusGained.Column.Hpsid)
Trace ('HpsID = ' ,MclbFocusGained.Hpsid)
Trace ('Source HpsID = ' ,DMclbFocusGained.Source.Hpsid)
Trace ('PhysicalIndex = ' ,MclbFocusGained.PhysicalIndex)
Trace ('ColumnIndex = ' , MclbFocusGained.ColumnIndex)
Trace ('VirtualIndex = ' , MclbFocusGained.VirtualIndex)
EndProc
```

CellFocusLost

This event is triggered when a table cell loses focus. Various properties of the event can be used to obtain information about the cell that lost focus.

Properties

The following table describes event properties for CellFocusLost.

CellFocusLost event properties

Property and Type	Description
Column:Column	This provides an object reference to the Column object that contains the cell clicked on. Properties and methods of the Column object can be called to obtain additional information or to perform operations.
ColumnIndex:Integer	This is the order number of the column, where the leftmost column is 1, the one to its right is 2, and so on. The numbering column (if present) is not included in the numbering. Thus if there is a numbering column, the column to its immediate right is 1.
HpsID:String	This returns the system identifier (HPSID) of the object that generated the event.
PhysicalIndex:Integer	This indicates the index (or occurrence number) of the row that contains the cell in the occurring view to which the table is data-linked.
Source:GuiObject	This is a reference to the Table object that generated the event, typed as a GuiObject.
VirtualIndex:Integer	This indicates the virtual row number for the row that contains the cell.

Support

This event is *not* supported for thin (HTML) clients. It is only supported for Java (thick) clients.

Example: CellFocusLost Event

The following is an example of the syntax:

```
Proc for CellFocusLost object TestMclb (MclbFocusLost object type CellFocusLostEvent)
Trace ('Column HpsID = ' ,MclbFocusLost.Column.Hpsid)
Trace ('HpsID = ' ,MclbFocusLost.Hpsid)
Trace ('Source HpsID = ' ,MclbFocusLost.Source.Hpsid)
Trace ('PhysicalIndex = ' ,MclbFocusLost.PhysicalIndex)
Trace ('ColumnIndex = ' , MclbFocusLost.ColumnIndex)
Trace ('VirtualIndex = ' , MclbFocusLost.VirtualIndex)
EndProc
```

ChildRuleEnd

This event is triggered to a parent rule when a child rule terminates. (See [ParentRuleEnd](#).)
ChildRuleEnd is not supported in C#.

Properties

The following table describes event properties for ChildRuleEnd.

ChildRuleEnd event properties

Property and Type	Description
Instance:String	This returns the instance name of the child rule that ended.
LongName:String	This is the long name for the child rule.
OutputView:View	This returns the output view of the child rule. This view can be mapped to any other view.

Example: ChildRuleEnd Event

The following is an example of the syntax:

```
*> example of Rule ChildRuleEndEvent <*
proc for ChildRuleEnd object Rule_A
(evtChildRuleEnd object type ChildRuleEndEvent)
dcl
myChar char(32);
myInteger Integer;
View myView contains myChar, myInteger;
enddcl
trace("Instance: ",evtChildRuleEnd.Instance)
trace("LongName: ",evtChildRuleEnd.LongName)
set myView := evtChildRuleEnd.OutputView
trace("myChar: ",myChar)
trace("myInteger: ",char(myInteger))
endproc
```

The following is an example of the output view being mapped to another view:

```
proc aaa FOR ChildRuleEnd object CUST_DIS
(evtChildRuleEnd object type ChildRuleEndEvent)
set CUST_DIS.CUST_INFO :=
evtChildRuleEnd.OutputView
endproc
```

If the child rule was detached with the INSTANCE clause, as shown in this code:

```
use RULE <rulename> DETACH INSTANCE <instance name>
```

then the Instance property contains the specified instance name; otherwise, it contains the long name of the rule that ended.

Click

This event is triggered if the user clicks the mouse button when the mouse is over a user interface object. It also occurs in the following situations:

- The **Enter** key is pressed when the window has a default push button.
- The spacebar is pressed when a check box, radio button, or push button has the focus.
- The user presses the mnemonic key for a push button, radio button, or check box. The mnemonic is the underlined character in the text; pressing **Alt** and a mnemonic character is equivalent to clicking on the object with the mouse and thus triggers the Click event.
- The user clicks on a menu item or presses the accelerator key combination (if any) associated with the menu item.
- The user selects an item in a list box or combo box by pressing an arrow key.
- The user selects a cell in a table by clicking with the mouse button or (if already in the table) by pressing an arrow key.
- The Selected property of a radio button is set to *True*.
- The value of the Selected property of a check box is changed.
- Data is mapped to a field data-linked to a radio button or check box.

Clicking on a radio button changes the data link associated with it when it is changing state (selected from unselected or vice versa).

Clicking on check box changes the data link associated with it.

The Click event is not supported for thin client EditField.

Properties

The following table describes event properties for Click Event.

Click event properties

Property and Type	Description
Column:Column	When triggered by a table, this provides an object reference to the Column object that was clicked on (to which the object is data-linked). Properties and methods of the Column object can be called to obtain additional information or to perform operations.
ColumnIndex:Integer	When triggered by a table, this provides the order number (index) of the column that was clicked on, where the left-most column is 1, the one to its right is 2, and so on. The numbering column (if present) is not included in the numbering. Thus, if there is a numbering column, the column to its immediate right is 1.
HpsID:String	This returns the system identifier (HPSID) of the object that generated this event. If this event is triggered by a table cell, the HpsID property contains the system identifier (HPSID) of the table (not the cell).
PhysicalIndex:Integer	When triggered by a table, this provides the occurrence number (index) of the row in the occurring view clicked on (to which the object is data-linked).
Source:GuiObject	This is a reference to the table object that generated the event, typed as a GuiObject. This provides an object reference to the object that triggered the event. This reference can be used to manipulate the object. If this event is triggered by a table cell, the Source property returns an object reference to the table itself (not the cell). The reference has the type of GuiObject.
VirtualIndex:Integer	When triggered by a table, this provides the virtual row number (index) for the row that was clicked on.

If the event is triggered by a table cell, the [Column:Column](#), [ColumnIndex:Integer](#), [PhysicalIndex:Integer](#), and [VirtualIndex:Integer](#) properties contain information about the row and column of the cell that generated the event. If the event is not triggered by a table, these properties are not used.

Example: Click Events

This example shows a Click event procedure that responds to only one object, namely the check box whose system identifier (HPSID) is 'CHECK_1'.


```

//////////Click Event Procedure by a Single Object//////////
// Checkbox CHECK_1 Event Click
Proc for Click object CHECK_1 (CheckBoxClick object type ClickEvent)
Trace(' HpsID = ', CheckBoxClick.Hpsid)
Trace(' Source HpsID = ', CheckBoxClick.Source.HpsID)
EndProc
// Spreadsheet MCLB_1 Event Click
Proc for Click object MCLB_1 (MclbClick object type ClickEvent)
Trace ('Column = ', MclbClick.Column.Hpsid)
Trace ('HpsID = ', MclbClick.Hpsid)
Trace ('Source HpsID = ', MclbClick.Source.Hpsid)
Trace ('PhysicalIndex = ', MclbClick.PhysicalIndex)
Trace ('ColumnIndex = ', MclbClick.ColumnIndex)
Trace ('VirtualIndex = ', MclbClick.VirtualIndex)
EndProc

```

This example shows a Click event procedure that is triggered by all similar objects (such as push buttons) and demonstrates how the HpsID property on the ClickEvent parameter can be used to determine which button was pressed.

```

//////////Click Event Procedure by Type of the Object//////////
// Checkbox CHECK_1 Event Click
Proc for Click Type CheckBox (CheckBoxClick object type ClickEvent)
Trace(' HpsID = ', CheckBoxClick.Hpsid)
Trace(' Source HpsID = ', CheckBoxClick.Source.HpsID)
EndProc
// Spreadsheet MCLB_1 Event Click
Proc for Click Type Table (TableClick object type ClickEvent)
Trace ('Column = ', MclbClick.Column.Hpsid)
Trace ('HpsID = ', MclbClick.Hpsid)
Trace ('Source HpsID = ', MclbClick.Source.Hpsid)
Trace ('PhysicalIndex = ', MclbClick.PhysicalIndex)
Trace ('ColumnIndex = ', MclbClick.ColumnIndex)
Trace ('VirtualIndex = ', MclbClick.VirtualIndex)
EndProc

```

Close

The *Close* event is triggered when the user attempts to close a window using either the system exit or the shortcut key. By default, the system exit does nothing because many business applications do not want events created by the user to cause changes. So, to allow the system exit to close the window, you must define the procedure to explicitly terminate the window by calling the *Close* event. Then, when the system exit is clicked, the *Close* event is generated and the system terminates the window based on that event.

This event has no methods or properties.

This event is not supported in the thin client development.

Support

This event is *not* supported for thin (HTML) clients, only for Java (thick) clients.

Example: Close Events

```

*> example of Window CloseEvent <*
proc for Close object APPB_SS_MCLB
(evtClose object type CloseEvent)
*> normal for this to then trigger the terminate event <*
thisrule.Terminate
endproc

```

You can also define this procedure:

```

proc CloseProcedure for Close type window
  (evtClose object type CloseEvent)
  (You can add other business logic here, such as an if-then clause if you want to check anything before
  closing.)
  windowname.terminate
endproc
where CloseProcedure is a name you define for a procedure and windowname is the object name for the
window object.

```

CommError

The *CommError* event is used in conjunction with HPS_COMM_ERROR_RULE system rule. Refer to [Rule](#) for the information about rules. For further information on the HPS_COMM_ERROR_RULE, see the *Deploying Applications Guide*. CommError event is not supported in C#.

Properties

The following table describes event properties for CommError.

CommError event properties

Property	Type	Description
abort	Boolean	Enable this flag to terminate the application on this communication error. This overrides any other setting and the application exits. It is disabled by default and setting it to true enables it.
callCommErrorRule	Boolean	Enable this flag to invoke the HPS_COMM_ERROR_RULE on exit from this event. By default this is enabled and the rule is called.
Exception	Object	This is the exception object with all the details of the error. Refer to Exception Properties .
LocalErrorCode	Integer	Error code from the client side. This is set when there is an error in client-side processing or the communication error of the request or response.
Message	String	The error message as a string.
RemoteErrorCode	Integer	The error code from the server side.
RemoteRuleName	String	Name of the remote rule that failed to execute.

Exception Properties

The following table describes exception properties for CommError.

CommError Exception Properties

Name	Type	Description
callingRuleId	String	The client rule name
commErrorView	View	An AppBuilder view that has various fields listed below. This is useful to map data to other views for error processing.
commErrorCode	Integer	The error code
errorID	Integer	The error ID
loginName	String	The user ID if given
protocolErrorCode	Integer	The communications error code
serverError	Integer	The error code from the server
serviceNameId	String	The short name of the remote rule
targetMachine	String	The server host name
targetProtocol	String	The protocol used for communication

targetServer	String	The target sever ID
tranId	Integer	A unique transaction ID
viewLength	Integer	The view length
workstationId	String	The client host name

Example: CommError Window Event

Here is an example of the syntax:


```

*> example of Rule CommErrorEvent <*
proc for CommError Type Rule
(eCommErr object type CommErrorEvent)
*> Trace CommErrorEvent properties <*
trace('eCommErr.abort: ', eCommErr.abort)
trace('eCommErr.callCommErrorRule: ', eCommErr.callCommErrorRule)
trace('eCommErr.LocalErrorCode: ', eCommErr.LocalErrorCode)
trace('eCommErr.Message: ', eCommErr.Message)
trace('eCommErr.RemoteErrorCode: ', eCommErr.RemoteErrorCode)
trace('eCommErr.RemoteRuleName: ', eCommErr.RemoteRuleName)
*> Trace CommErrorException properties <*
trace('eCommErr.Exception.callingRuleId: ', eCommErr.Exception.callingRuleId)
trace('eCommErr.Exception.commErrorCode: ', eCommErr.Exception.commErrorCode)
trace('eCommErr.Exception.errorID: ', eCommErr.Exception.errorID)
trace('eCommErr.Exception.loginName: ', eCommErr.Exception.loginName)
trace('eCommErr.Exception.protocolErrorCode: ', eCommErr.Exception.protocolErrorCode)
trace('eCommErr.Exception.serverError: ', eCommErr.Exception.serverError)
trace('eCommErr.Exception.serviceNameId: ', eCommErr.Exception.serviceNameId)
trace('eCommErr.Exception.targetMachine: ', eCommErr.Exception.targetMachine)
trace('eCommErr.Exception.targetProtocol: ', eCommErr.Exception.targetProtocol)
trace('eCommErr.Exception.targetServer: ', eCommErr.Exception.targetServer)
trace('eCommErr.Exception.tranID: ', eCommErr.Exception.tranID)
trace('eCommErr.Exception.viewLength: ', eCommErr.Exception.viewLength)
trace('eCommErr.Exception.workstationId: ', eCommErr.Exception.workstationId)
endproc

```

Converse

The *Converse* event is triggered when a user interface action occurs that would have caused the converse window statement to return in legacy applications. This event is triggered by the window, not by other user interface objects.

 This event is used to provide backwards compatibility and should not be used for new applications.

To use the *Converse* event to port existing applications, define a *Converse* event procedure, and include the logic that followed the converse window statement in the converse window loop. The *Converse* event object that is passed to the *Converse* event procedure has a number of read-only properties that make available the same information provided in the predefined system view HPS_EVENT_VIEW. If HPS_EVENT_VIEW is attached to the rule in the application hierarchy, then HPS_EVENT_VIEW is updated with the appropriate information before the *Converse* event is triggered. Your application code can then obtain the information it needs from the properties of the *Converse* event or from HPS_EVENT_VIEW.

Properties

The following table describes event properties for Converse.

Converse event properties

Property and Type	Description
EventParam() :String	This returns a string that contains the name of the event, such as 'HPS_PB_CLICK', 'HPS_IMMEDIATE_RETURN', and 'HPS_MENU_SELECT'.

EventQualifier() :String	
EventSource() :String	This returns the system identifier (HPSID) of the user interface object on which the action occurred which triggered the event.
EventType() :Integer	
EventView() :String	

EventParam() :String

This returns an optional, event-specific string that contains additional information. This string can be empty.

EventQualifier() :String

This returns an optional, event-specific string that contains additional information. This string can be empty.

EventSource() :String

This provides the system identifier (HPSID) of the user interface object on which the action occurred that triggered the event. For example, for an HPS_PB_CLICK event, this property provides the system identifier (HPSID) of the push button that was clicked.

EventType() :Integer

This returns the type of event of the predefined system view HPS_EVENT_VIEW, as one of the following values (as defined in the Constants class):

- SYSTEM_EVENT
- INTERFACE_EVENT
- USER_EVENT
- ASYNC_EVENT
- LANDP_EVENT
- LANDP_REQUEST_EVENT
- LANDP_SYSTEM_EVENT

EventView() :String

This returns the name of an optional, event-specific view that contains additional information. This string is empty if no view is provided.

[Example: Converse Event](#)

```

*> example of Window ConverseEvent <*>
proc for Converse type Window
  (evtConverse object type ConverseEvent)
  trace('evtConverse.EventParam: ', evtConverse.EventParam)
  trace('evtConverse.EventQualifier: ', evtConverse.EventQualifer)
  trace('evtConverse.EventSource: ', evtConverse.EventSource)
  trace('evtConverse.EventType: ', evtConverse.EventType)
  trace('evtConverse.EventView: ', evtConverse.EventView)
endproc

```

DataRequired

The *DataRequired* event is triggered by the [Table](#) object when it requires different data mapped into its data-linked occurring view. It needs this information to respond to the user's request to scroll data. This event is used to implement the [Smooth Scrolling in a DataRequired Event](#). It can be used to instruct the table to automatically update its display and to specify to the table what data has been placed in the occurring view.

Smooth Scrolling in a DataRequired Event

The Smooth Scrolling option scrolls the page from one link to another rather than jumping to it directly. This can prevent user disorientation, particularly in a large document. There are two ways to implement smooth scrolling. The first approach is similar to that used in previous versions

of the product; it requires you to explicitly call the `setVirtualListBoxSize()` method, and optionally the `setFirstVisibleRow()` method. In another more efficient approach, the table calls these methods automatically.

Both of these approaches share the requirement that the program logic must obtain the appropriate data from the data source and place it in the data-linked occurring view. The program logic knows which data is needed by the `TopVirtualRow` property of `DataRequired` event. It requires you to explicitly call the `setVirtualListBoxSize()` method, and, optionally, the `setFirstVisibleRow()` method. In the second approach, these methods are called automatically by the table itself. In both approaches, the `DataRequired` event procedure must fetch the appropriate data and place it in the data-linked occurring view.

In the first approach, when `DataRequired` event is triggered, its event procedure calls the table's `ElevatorPosition()` method to determine which virtual row should be placed in the top of the occurring view. Once the data has been fetched and is placed in the occurring view, the table's `setVirtualListBoxSize()` method is called to specify what virtual row was actually placed in the top of the occurring view and the total number of virtual rows in the data source. Optionally, the table's `setFirstVisibleRow()` method can be called to specify that a row other than that at the top of the occurring view should be displayed at the top of the table. The table is then updated with the appropriate data.

A more efficient way to implement smooth scrolling is to make use of the `TopVirtualRow` and `Refresh` properties of the `DataRequired` event. If, after fetching the data into the data-linked occurring view, the event procedure sets the event's `Refresh` property to `True`, then there is no need for the event procedure to call `setVirtualListBoxSize()` and `setFirstVisibleRow()` ?these methods are automatically called by the table itself. When this event is triggered, the `TopVirtualRow` property is initialized with the virtual row index of the data that needs to be displayed at the top of the table. At this point, the event procedure for the event fetches the data and places it into the data-linked occurring view.

There is another way to implement smooth scrolling that is more compatible with previous versions of the product. In this approach, when this event is triggered, its event procedure calls the table's `ElevatorPosition()` method to determine which virtual row should be placed at the top of the occurring view. Once the data has been fetched and is placed in the occurring view, the table's `setVirtualListBoxSize()` method is called to specify what virtual row was actually placed in the top of the occurring view and the total number of virtual rows in the data source. The table is then updated with the appropriate data.

For an example of smooth scrolling using system components, refer to the section on smooth scrolling in *System Components Reference Guide* .

Properties

The following table describes event properties for `DataRequired`.

DataRequired event properties

Property and Type	Description
Direction():Integer	This is a read only property representing the scrolling direction. It can have one of the following values: Constants.UP, Constants.DOWN.
HpsID:String	This returns the system identifier (HPSID) of the object that generated the event.
Refresh():Boolean	This indicates whether the table should automatically update its display.
Source:GuiObject	This is a reference to the object that generated the event, typed as a GuiObject.
TopVirtualRow:Integer	On input, this contains the virtual index of the row that must be displayed at the top of the table. On output, this must contain the virtual index of the first row in the data-linked occurring view. (Output value is significant only if Refresh is True.)
TypeString:String	This returns a string to indicate the type of event.

The [HpsID:String](#) property of `DataRequired` event specifies the system identifier (HPSID) of the table which is requesting data. The `TypeString` property specifies the kind of action that resulted in the need for additional data.

The [Refresh\(\):Boolean](#) property indicates if the table should automatically update the new top virtual row. If you want a new top row other than the current one, set the event property `TopVirtualRow` before setting this method. If `Refresh` is `True`, the table updates the new top row automatically by calling `setVirtualListBoxSize` and `FirstVisibleRow` when the `DataRequired` event procedure exits. If `Refresh` is `False`, you must include code, within the `DataRequired` event procedure, to cause the display to be updated. This code consists of a call to the table's `setVirtualListBoxSize()` method and, optionally, the `setFirstVisibleRow()` method. By default, `Refresh` is set to `False`. We recommend that you allow the table to automatically update these values.

When the event procedure is first invoked, the [TopVirtualRow:Integer](#) property is pre-initialized with the virtual index of the row that must be displayed at the top of the table. When the event procedure exits, `TopVirtualRow` must contain the virtual index of the first row in the data-linked occurring view if `Refresh` is set to `True`.

If the application does not use a back buffer to include rows at the top of the occurring view that are before the row to be displayed at the top of the displayed table, then the input and output values of `TopVirtualRow` are identical. However, if a back buffer is used, then the input and output values are different.

If the virtual table size (which determines the position of the thumb in the table's vertical scroll bar) is considered to change as a result of the data fetch, the table's `setVirtualListBoxSize()` method must be called explicitly in the event procedure, even if `Refresh` is `True`.

The `TopVirtualRow` property has different meanings on input and output. On input, when the event procedure is first invoked, `TopVirtualRow` is pre-initialized with the virtual index of the row that must be displayed at the top of the table. The output value of `TopVirtualRow` is significant only if `Refresh` is `True` when the event procedure exits. If it is set to `True`, then on output `TopVirtualRow` must contain the virtual index of the first row in the data-linked occurring view.

Refresh():Boolean

This indicates whether the table should automatically update its display.

Direction():Integer


This is a read only property representing the scrolling direction. It can have one of the following values: Constants.UP or Constants.DOWN.

Example: Table DataRequiredEvent

```
*> example of Table DataRequiredEvent <*
proc for DataRequired object OBJ_LB
(e object type DataRequiredEvent)
endproc
```

DoubleClick

The *DoubleClick* event is triggered if the user double-clicks the mouse button when the mouse is over a user interface object. It is triggered by most common user interface objects, with the notable exception of check boxes and push buttons. The DoubleClick event is not supported for thin client EditField.

 Double-clicking on an object also triggers ClickEvent, if ClickEvent is defined for that object. The order of events generated are ClickEvent, DoubleClickEvent.

Properties

The following table describes properties for DoubleClick.

DoubleClick properties

Property and Type	Description
Column:Column	When triggered by a table, this provides an object reference to the Column object that was clicked on (to which the object is data-linked). Properties and methods of the Column object can be called to obtain additional information or to perform operations.
ColumnIndex:Integer	When triggered by a table, this provides the order number (index) of the column that was clicked on, where the leftmost column is 1, the one to its right is 2, and so on. The numbering column (if present) is not included in the numbering. Thus if there is a numbering column, the column to its immediate right is 1.
HpsID:String	This returns the system identifier (HPSID) of the object that generated the event.
PhysicalIndex:Integer	When triggered by a table, this provides the occurrence number (index) of the row in the occurring view clicked on (to which the object is data-linked).
Source:GuiObject	This is a reference to the table object that generated the event, typed as a GuiObject.
VirtualIndex:Integer	When triggered by a table, this provides the virtual row number (index) for the row that was clicked on.

If the event is triggered by a table cell, the [Column:Column](#), [ColumnIndex:Integer](#), [PhysicalIndex:Integer](#), and [VirtualIndex:Integer](#) properties contain information about the row and column of the cell that generated the event. If the event is not triggered by a table, these properties are not used.

The [HpsID:String](#) property of the event can be used to obtain the system identifier (HPSID) of the object that generated the Click event. If this event is triggered by a table cell, the HpsID property contains the system identifier (HPSID) of the table (not the cell).

The [Source:GuiObject](#) property provides an object reference to the object that triggered the event. This reference can be used to manipulate the object. If this event is triggered by a table cell, the Source property returns an object reference to the table itself (not the cell). The reference has the type of GuiObject.

Example: DoubleClickEvent

```

//////////DOUBLECLICK Event Procedure by Type of the Object//////////
// Spreadsheet MCLB_1 Event DOUBLECLICK
Proc for DoubleClick Type Table (TableDoubleClick object type DoubleClickEvent)
Trace ('Column = ', TableDoubleClick.Column.Hpsid)
Trace ('HpsID = ', TableDoubleClick.Hpsid)
Trace ('Source HpsID = ', TableDoubleClick.Source.Hpsid)
Trace ('PhysicalIndex = ', TableDoubleClick.PhysicalIndex)
Trace ('ColumnIndex = ', TableDoubleClick.ColumnIndex)
Trace ('VirtualIndex = ', TableDoubleClick.VirtualIndex)
EndProc
// Multilineedit MLE_1 Event DOUBLECLICK
Proc for DoubleClick Type MultiLineEdit (MultiLineEditDoubleClick object type DoubleClickEvent)
Trace(' HpsID = ', MultiLineEditDoubleClick.Hpsid)
Trace('Source HpsID = ', MultiLineEditDoubleClick.Source.HpsID)
EndProc
//////////DOUBLECLICK Event Procedure by Every Single Object//////////
// Spreadsheet MCLB_1 Event DOUBLECLICK
Proc for DoubleClick object MCLB_1 (MclbDoubleClick object type DoubleClickEvent)
Trace ('Column = ', MclbDoubleClick.Column.Hpsid)
Trace ('HpsID = ', MclbDoubleClick.Hpsid)
Trace ('Source HpsID = ', MclbDoubleClick.Source.Hpsid)
Trace ('PhysicalIndex = ', MclbDoubleClick.PhysicalIndex)
Trace ('ColumnIndex = ', MclbDoubleClick.ColumnIndex)
Trace ('VirtualIndex = ', MclbDoubleClick.VirtualIndex)
EndProc
// Multilineedit MLE_1 Event DOUBLECLICK
Proc for DoubleClick object MLE_1 (MultiLineEditDoubleClick object type DoubleClickEvent)
Trace(' HpsID = ', MultiLineEditDoubleClick.Hpsid)
Trace('Source HpsID = ', MultiLineEditDoubleClick.Source.HpsID)
EndProc

```

FieldError

The *FieldError* event is triggered when an editable field loses focus or when the Enter key is pressed while the field has focus, and the data in the field is in error. It allows the application to specify whether the data should be rolled back to the last known acceptable value and whether a message box describing the data should be displayed.

This event is triggered by the following objects:

- [ComboBox](#)
- [EditField](#)
- [MultiLineEdit](#)
- [PasswordField](#)
- [Table](#)

To specify whether the data should be rolled back, set the [Rollback:Boolean](#) property. To specify whether a message box should be shown if the field is not rolled back, set the [ShowMessage:Boolean](#) property. By default, the data is *not* rolled back and an error message is *not* shown. A setting in the VALIDATION section of APPBUILDER.INI, SHOW_FIELD_ERROR_MESSAGE_BOX_DEFAULT, can be used to configure the default for *ShowMessage*.

For more information, see [Field-level Validation](#).

Support

This event is *not* supported for thin (HTML) clients.

Properties

The following table describes event properties for FieldError.

FieldError event properties

Property and Type	Description
ColumnIndex:Integer	When triggered by a table, this provides the order number (index) of the column that was clicked on, where the left-most column is 1, the one to its right is 2, and so on. The numbering column (if present) is not included in the numbering. Thus, if there is a numbering column, the column to its immediate right is 1.
HpsID:String	Read-only property. This returns the system identifier (HPSID) of the object that generated the event.

PhysicalIndex:Integer	When triggered by a table, this provides the occurrence number (index) of the row in the occurring view clicked on (to which the object is data-linked).
Rollback:Boolean	This indicates whether the data should be rolled back to the last known acceptable value. (The default value is FALSE.)
ShowMessage:Boolean	This indicates whether a message box describing the error should be shown if the data is not rolled back. (The default value is FALSE; unless the default is being specified by a setting in appbuilder.ini.)
Source:GuiObject	Read-only property. This is a reference to the object that generated the event, typed as a GuiObject.
VirtualIndex:Integer	When triggered by a table, this provides the virtual row number (index) for the row that was clicked on.

If this event is triggered by a table cell, the [HpsID:String](#) property contains the system identifier (HPSID) of the table itself (not the cell), and the Source property returns an object reference to the table.

Example: FieldError Event

```
// FieldErrorEvent for EditField
Proc for FieldError object EDIT_1(FieldErrorEdit object type FieldErrorEvent)
Trace('HpsID = ', FieldErrorEdit.HpsID)
Trace('RollBack =', FieldErrorEdit.RollBack)
Trace('ShowMessage =', FieldErrorEdit.ShowMessage)
Trace('Source HpsID =', FieldErrorEdit.Source.Hpsid)
endproc
//FieldErrorEvent For MCLB
Proc for FieldError object MCLB_1(FieldErrorMclb object type FieldErrorEvent)
Trace('HpsID = ', FieldErrorMclb.HpsID)
Trace('RollBack =', FieldErrorMclb.RollBack)
Trace('ShowMessage =', FieldErrorMclb.ShowMessage)
Trace('Source HpsID =', FieldErrorMclb.Source.Hpsid)
Trace('ColumnIndex =', FieldErrorMclb.ColumnIndex)
Trace('PhysicalIndex =', FieldErrorMclb.PhysicalIndex)
Trace('VirtualIndex =', FieldErrorMclb.VirtualIndex)
endproc
```

FieldValidation

The *FieldValidation* event allows an application to validate the data in an editable field when the field loses focus or when the **Enter** key is pressed while the field has focus. It is triggered only if there is no intrinsic data or formatting errors. If there are errors, the [FieldError](#) event is triggered.

This event is triggered by the following objects:

- [ComboBox](#)
- [EditField](#)
- [MultilineEdit](#)
- [PasswordField](#)
- [Table](#)

For more information, see [Field-level Validation](#).

Usage

This event allows the application to specify whether the data should be accepted, rolled back to the last known acceptable value, or considered in error. The default value is *ACCEPT*.

This event also allows the application to specify whether a message box should be displayed if the data is regarded as "in error". By default, a message box is shown if the data is in error.

To specify whether the data should be accepted, rolled back, or considered in error, set the [Response:Integer](#) property to one of the following values:

- *ACCEPT*
- *ROLLBACK*
- *IN_ERROR*

To specify whether a message box should be shown if the field is in error, set the [ShowMessage:Boolean](#) property to *True* or *False*.

If this event is triggered by a table cell, the [HpsID:String](#) property contains the system identifier (HPSID) of the table itself (not the cell), and the Source property returns an object reference to the table.

Properties

The following table describes event properties for FieldValidation.

FieldValidation event properties

Property and Type	Description
ColumnIndex:Integer	When triggered by a table, this provides the order number (index) of the column that was clicked on, where the left-most column is 1, the one to its right is 2, and so on. The numbering column (if present) is not included in the numbering. Thus, if there is a numbering column, the column to its immediate right is 1.
HpsID:String	Read-only property. This returns the system identifier (HPSID) of the object that generated the event.
PhysicalIndex:Integer	When triggered by a table, this provides the occurrence number (index) of the row in the occurring view clicked on (to which the object is data-linked).
Response:Integer	This indicates whether the data should be accepted, rolled back to the last known acceptable value, or regarded in error. The default value is accepted.
ShowMessage:Boolean	This indicates whether a message box describing the error should be shown if the data is not rolled back. (The default value is TRUE.)
Source:GuiObject	Read-only property. This is a reference to the object that generated the event, typed as a GuiObject.
VirtualIndex:Integer	When triggered by a table, this provides the virtual row number (index) for the row that was clicked on.

Support

This event is *not* supported for thin (HTML) clients.

Example: FieldValidation

```
*> example of Field FieldValidationEvent <*
// FieldValidationEvent for EditField
Proc for FieldValidation object EDIT_1(FieldValidationEdit object type FieldValidation)
Trace('HpsID = ', FieldValidationEdit.HpsID)
Trace('Response = ', FieldValidationEdit.Response)
Trace('ShowMessage = ', FieldValidationEdit.ShowMessage)
Trace('Source HpsID = ', FieldValidationEdit.Source.Hpsid)
endproc
//FieldValidationEvent For MCLB
Proc for FieldValidation object MCLB_1(FieldValidationMclb object type FieldValidationEvent)
Trace('HpsID = ', FieldValidationMclb.HpsID)
Trace('Response = ', FieldValidationMclb.Response)
Trace('ShowMessage = ', FieldValidationMclb.ShowMessage)
Trace('Source HpsID = ', FieldValidationMclb.Source.Hpsid)
Trace('ColumnIndex = ', FieldValidationMclb.ColumnIndex)
Trace('PhysicalIndex = ', FieldValidationMclb.PhysicalIndex)
Trace('VirtualIndex = ', FieldValidationMclb.VirtualIndex)
endproc
```

FocusGained

The *FocusGained* event is triggered when a user interface object gains focus. Tables trigger the *CellFocusGained* event. They do *not* trigger the *FocusGained* event. The *FocusGained* event is not supported for thin client EditField.

Properties

The following table describes event properties for FocusGained.

FocusGained event properties

Property and Type	Description
-------------------	-------------

HpsID:String	This returns the system identifier (HPSID) of the object that generated the event.
Source:GuiObject	This is a reference to the object that generated the event, typed as a GuiObject.

The [HpsID:String](#) property of the event can be used to obtain the system identifier (HPSID) of the object that gained focus. The [Source:GuiObject](#) property provides a reference to the object that triggered the event in the form of a GuiObject and can be used to manipulate the object.

Support

This event is *not* supported for thin (HTML) clients. It is only supported for Java (thick) clients.

Example: FocusGained

```
*> example of Window FocusGainedEvent <*
// FocusGainedEvent for EditField
Proc for FocusGained object EDIT_1(FieldFocusGained object type FocusGainedEvent)
Trace('HpsID = ', FieldFocusGained.HpsID)
Trace('Source HpsID = ', FieldFocusGained.Source.Hpsid)
endproc
```

FocusLost

The *FocusLost* event is triggered when a user interface object loses focus. If the focus is removed from this field temporarily (for example, because another window is activated) this event is not triggered. Tables do *not* trigger this event but rather the [CellFocusLost](#) event. The FocusLost event is not supported for thin client EditField.

Properties

The following table describes event properties for FocusLost.

FocusLost event properties

Property and Type	Description
HpsID:String	This returns the system identifier (HPSID) of the object that generated the event.
Source:GuiObject	This is a reference to the object that generated the event, typed as a GuiObject.

The [HpsID:String](#) property of the event can be used to obtain the system identifier (HPSID) of the object that lost focus. The [Source:GuiObject](#) property provides a reference to the object that triggered the event in the form of a GuiObject and can be used to manipulate the object.

Support

This event is *not* supported for thin (HTML) clients, but is supported for Java (thick) clients.

Example: FocusLostEvent

```
*> example of Window FocusLostEvent <*
// FocusLostEvent for EditField
Proc for FocusLost object EDIT_1(FieldFocusLost object type FocusLostEvent)
Trace('HpsID = ', FieldFocusLost.HpsID)
Trace('Source HpsID = ', FieldFocusLost.Source.Hpsid)
endproc
```

HeaderClick

This event is generated when you click on a table column header. The table's data can be sorted depending on the column index, if the property UsingDefaultSorting is TRUE (the sorting order can be one of Constants.ASCENDING or Constants.DECENDING, the default is the former).

The following example will sort the table data in DESCENDING order based on the column index:

```
proc for HeaderClick object MCLB_1(e object pointer to HeaderClickEvent)
set e.UsingDefaultSorting := true
set e.SortingOrder := Constants.DECENDING
endproc
```

The sorting of Virtual Table is undefined, the smooth scrolling will reset the sorting when DataRequiredEvent is generated. You are advised not to use default sorting with virtual table sizes(VirtualListBoxSize).

Properties

The following table describes event properties for HeaderClick.

HeaderClick properties

Properties	Description
Column:Column	The Column object of the table header being clicked on.
ColumnIndex:Integer	column index of the column
HpsID:String	HpsID of the table
SortingOrder:Integer	order by which the mclb to be sorted if the property UsingDefaultSorting is true
Source:GuiObject	source at generated the event
UsingDefaultSorting:Boolean	When this property is true, the table is sorted depending on the property SortingOrder.

Initialize (for Rule)

The rule *Initialize* event is triggered on entry to a rule each time it is called. If the rule has a window attached, the window *Initialize* event is triggered after this event. This event is an ideal location to place code for initialization of data or state information.



Do not make any assumptions regarding the status of the window in the Initialize (for rule) event.

Rule Initialize Event Properties

Property and Type	Description
SourceName:String	Returns the name of the Rule object.

Example: Rule Initialize Event

```
*> example of Rule InitializeEvent <*
proc for Initialize type Rule
(eRuleInit object type InitializeEvent)
trace('eRuleInit.SourceName: ', eRuleInit.SourceName)
endproc
```

Initialize (for Window)

The window *Initialize* event is triggered after the window and all controls have been created, and after the rule *Initialize* event. At the time of this event, while the window exists, it will not be visible. The window will automatically be made visible after this event has occurred. This event is an ideal location to place code to initialize data structures and call ObjectSpeak methods that set the appearance or behavior of user interface objects.

Window Initialize Event Properties

Property and Type	Description
SourceName:String	Returns the name of the Window object.

[Example: Window Initialize Event](#)

```
*> example of Window InitializeEvent <*
proc for Initialize type Window
(eWindowInit object type InitializeEvent)
trace('eWindowInit.SourceName: ', eWindowInit.SourceName)
endproc
```

Here is another example of the syntax:

```
proc Initialize for Initialize object MAIN_WINDOW
(e object type InitializeEvent)
*> initialize data <*
map 0 to NumOfOrders
*> call ObjectSpeak methods to modify window <*
NameField.setForeground(Color.RED)
CustomerIDField.setEditable(False)
endproc
```

MessageBox

The MessageBox event is supported in the thin client only. When a *showMessageBox* method is invoked, a JavaScript message box is displayed on the Browser screen. Two message types are supported: ERROR and QUESTION. If the message type is ERROR, the message is displayed, and after acknowledging the message by clicking **OK**, you can continue with the current page. If the message type is QUESTION, the response (YES/NO or OK/CANCEL) is sent back to the Server and fired as a MessageBox on the Window.

```
*> example of MessageBoxEvent <*
proc for MessageBox object WINDOW_A
(e object type MessageBoxEvent)
*> e.response will be Constants.OK, Constants.CANCEL, Constants.YES or Constants.NO
<*
caseof (e.response)
case Constants.OK
endcase
endproc
```

The Message Box is asynchronous, meaning that it will not be displayed until the currently active event procedure has completed.

PageSelect

The *PageSelect* event is fired when the user selects a tab page.

Properties

The following table describes event properties for PageSelect.

PageSelect properties:

Property and Type	Description
Source:GuiObject	Returns the source object (TabControl).
HpsID:String	Returns the system identifier (HPSID) of the tab control.
SelectedPageIndex:Integer	Returns the index of the selected tab page.

ParentRuleEnd

The *ParentRuleEnd* event is triggered on a detached rule when the parent rule ends. See [ChildRuleEnd](#). ParentRuleEnd is not supported in C#.

Properties

The following table describes event properties for ParentRuleEnd.

ParentRuleEnd event properties

Property and Type	Description
TerminateChild():Boolean	

TerminateChild() :Boolean

Child rules are terminated along with parent rules by default as specified by the CLOSE_DETACHED_RULES_WITH_PARENT setting in the APPBUILDER.INI file. This property overrides the default from the APPBUILDER.INI file setting.

Example: ParentRuleEnd

```
*> example of Rule ParentRuleEvent <*
proc for ParentRuleEnd type Rule
(eParentRuleEnd object type ParentRuleEndEvent)
trace('eParentRuleEnd.TerminateChild: ',
eParentRuleEnd.TerminateChild)
endproc
```

Post

The *Post* event is triggered when a rule uses the *Post* method to post a view to another rule. The event is triggered on the rule that receives the posted view. Post event is not supported in C#.

Properties

The following table describes event properties for Post.

Post event properties

Property and Type	Description
Internal():Boolean	
LongName:String	This is the long name for the posting rule.
SourceName():String	This returns the name of the rule that posted the view (the source rule)
SourceObject():Object	
Subject():String	
View():View	This returns a reference to the view that was posted. The view returned by the View property is an untyped view. This means that it must be mapped either to a view defined in the hierarchy or to a view defined locally in the rule, before its fields can be accessed.
ViewName	This returns the name of the view that was posted.

Internal():Boolean

This determines whether the event originates from an AppBundle rule.

SourceName():String

This is the instance name of event source.

SourceObject():Object

This is the event source.

Subject():String

This is the event name.

View():View

This is a reference to a view.

[Example: PostEvent](#)

```
*> example of Rule PostEvent <*
proc for Post type Rule
(evtPost object type PostEvent)
trace('evtPost.Internal: ',evtPost.Internal)
trace('evtPost.LongName: ',evtPost.LongName)
trace('evtPost.SourceName: ',evtPost.SourceName)
trace('evtPost.Subject: ',evtPost.Subject)
trace('evtPost.View: ',evtPost.View)
trace('evtPost.ViewName: ',evtPost.ViewName)
```

RuleEnd

The *RuleEnd* event provides notification that a rule has ended. This event is never raised in C#.

Properties

The following table describes event properties for RuleEnd.

RuleEnd event properties

Property and Type	Description
Instance:String	This returns the instance name of the rule that ended. If the rule was detached with the INSTANCE clause, (as USE RULE <rulename> DETACH INSTANCE <instance name>) then the Instance property contains the specified instance name; otherwise, it contains the long name of the rule that ended.
LongName:String	This is the long name for the calling rule.
OutputView:View	This returns the output view of the rule that ended. This view can be mapped to any other view.

[Example: RuleEnd Event](#)

```
*> example of Window RuleEndEvent <*
proc for RuleEnd type Rule
(eRuleEnd object type RuleEndEvent)
trace('eRuleEnd.Instance: ', eRuleEnd.Instance)
trace('eRuleEnd.LongName: ', eRuleEnd.LongName)
trace('eRuleEnd.OutputView: ', eRuleEnd.OutputView)
endproc
```

Here is an example of a *RuleEnd* event for the OutputView property:

```

proc aaa for RuleEnd object CUST_DIS
(e object type RuleEndEvent)
map e.OutputView to CUST_INFO of CUST_DIS
endproc

```

The rule that ends must have been detached from the rule that receives the event, using the following syntax:
 use RULE < rule_name > DETACH < rule_object_name >
 where < rule_object_name > is the name of the local variable (defined in the dcl section) that references the rule.

SQLError

This event provides SQL error information. In order for this event to be enabled, the Rule must have its "DMBS Usage" property enabled. This also enables the rule access to the AppBuilder SQLCA system view which may provide additional information about the SQL error. SQLError event is not supported in C#.

Properties

The following table describes event properties for SQLError.

SQLError event properties

Property and Type	Description
Details:View	View containing the following properties
ErrorCode:Integer	The SQL code as implemented by the SQL provider
Message:String	The SQL message as implemented by the SQL provider

Details:View

This is a view which owns fields related to the SQL error.

ErrorCode:Integer

This is the SQL error code for the event.

Message:String

This is a string representing the SQL message as implemented by the SQL provider.

[Example: SQLError Event](#)

```

*> example of Rule SQLErrorEvent < *
proc for SQLError type Rule
(eSQLError object type SQLErrorEvent)
trace('eSQLError.Details.ErrorCode: ', eSQLError.Details.ErrorCode)
trace('eSQLError.Details.SqlState: ', eSQLError.Details.SqlState)
endproc

```

Terminate (for Rule)

The rule *Terminate* event is triggered just prior to the rule exiting. If the rule has a window, this event will be triggered after the window *Terminate* event.



Do not make any assumptions regarding the status of the window in the rule Terminate event.

Properties

The following table describes event properties for Terminate (forRule).

Rule Terminate event properties

Property and Type	Description
SourceName:String	Returns the name of the Rule object.

Example: Rule Terminate Event

```
*> example of rule TerminateEvent <*
proc for Terminate type Rule
(eRuleTerminate object type TerminateEvent)
trace('eRuleTerminate.SourceName: ', eRuleTerminate.SourceName)
endproc
```

Terminate (for Window)

The Window *Terminate* event is triggered on the window closing. The rule *Terminate* event is triggered after this event.

Properties

The following table describes event properties for Terminate (for Window).

Window Terminate event properties

Property and Type	Description
SourceName:String	Returns the name of the Window object.

Example: Window Terminate Event

```
*> example of TerminateEvent <*
proc for Terminate type Window
(eWindowTerminate object type TerminateEvent)
trace('eWindowTerminate.SourceName: ', eWindowTerminate.SourceName)
endproc
```

Timer

The *Timer* event is triggered by the [Timer](#) object to provide one or more timed notifications at regular intervals. *Timer* events are normally used to either delay performing an action for a specified period of time or to repeat an action, such as updating a display field.

Properties

The following table describes event properties for Timer event.

Timer event properties

Property and Type	Description
HpsID:String	This returns the system identifier (HPSID) of the timer object that generated the event.
Source:GuiObject	This is a reference to the object that generated the event, typed as a GuiObject.

Example: Timer Event


```

*> example of Timer object TimerEvent <*
proc for Timer object myTimer
(eTimer object type TimerEvent)
trace('eTimer.HpsID: ', eTimer.HpsID)
trace('eTimer.Source.Enabled: ', eTimer.Source.Enabled)
trace('evtTimer.Source.Focus: ', eTimer.Source.Focus)
endproc

```

WindowError

The *WindowError* event is triggered when window validation fails either because one or more fields contain errors or because mandatory fields do not contain any data. Window validation occurs when a push button or menu item with the *Validate* property set to *True* is activated.

For more information, refer to [Window-level Validation](#).

This event is not supported in the thin client development, neither in C#.

Properties

The following table describes event properties for WindowError.

WindowError event properties

Property and Type	Description
FieldError():Boolean	If this event was triggered because a field contains an error, this property is True.
HpsID:String	This returns the short name of the window that generated the event, since the window does not have a system identifier (HPSID).
MandatoryError():Boolean	If a mandatory field has no data, then this property is True.
ShowMessage:Boolean	This indicates whether a message box describing the error should be shown if the data is not rolled back. By default, a message box is shown; the default value is TRUE.
Source:GuiObject	This provides an object reference to the object that triggered the event ? in this case, the window itself. This reference, which has the type of GuiObject, can be used to manipulate the window.

For backwards compatibility with previous versions of this product, the [IgnoreValidation:Boolean](#) property on [PushButton](#) objects and [MenuItem](#) objects can be set to *False* in order to trigger window validation. Legacy applications can suppress the check on mandatory fields by setting the [CheckMandatoryFields:Boolean](#) property to *False* in those objects.

MandatoryError():Boolean

This indicates whether window validation failed because mandatory fields are empty.

FieldError():Boolean

This indicates whether window validation failed because one or more fields are in error.

Example:WindowError Event

```

*> example of WindowErrorEvent <*
proc for WindowError object OSPK_EVT_WINERR_WND
(WinErr object type WindowErrorEvent)
Trace (' HpsID =', WinErr.HpsId)
Trace (' Source HpsID =', WinErr.Source.HpsId)
Trace (' FieldError =', WinErr.FieldError)
Trace (' MandatoryError =', WinErr.MandatoryError)
Trace (' ShowMessage =', WinErr.ShowMessage)
endproc

```

WindowValidation

The *WindowValidation* event allows an application to verify that fields in the window have acceptable data. It is triggered when the user clicks a

push button or menu item whose Validate property is set to *True* , no fields are in error, and all mandatory fields contain data. If there are errors, the [WindowError](#) event is triggered.

This event allows the application to validate the window data as a whole and prevent the window from closing if the data is unacceptable. For example, the application can use this event to verify that the data in each field is consistent with data in other fields.

For more information, refer to [Window-level Validation](#).

This event is not supported in the thin client development.

Properties

The following table describes event properties for WindowValidation.

WindowValidation event properties

Property and Type	Description
Accept():Boolean	This determines whether the fields in the window have acceptable data.
HpsID:String	This returns the short name of the window that generated the event, since the window does not have a system identifier (HPSID).
ShowMessage:Boolean	This indicates whether a message box should be shown if the data is not accepted. The default value is TRUE.
Source:GuiObject	This provides a reference to the object that triggered the event ? in this case, the window itself. The reference has the type GuiObject and can be used to manipulate the window.

For backwards compatibility with previous versions of this product, the [IgnoreValidation:Boolean](#) property on [PushButton](#) objects and [MenuItem](#) objects can be set to *False* in order to trigger window validation. Legacy applications can suppress the check on mandatory fields by setting the [CheckMandatoryFields:Boolean](#) property to *False* in those objects.

This event also allows you to specify whether a message box should be displayed if the data is not acceptable. By default, the message box is shown. To prevent the message box display, set [ShowMessage:Boolean](#) to *False* .

Accept():Boolean

This specifies whether the data in the window is acceptable. The default value is *True* .

To specify that the data in the window is not acceptable, set the event Accept property to *False* . If it is *True* , then the push button or menu item that triggered window validation triggers the [Click](#) event. If it is *False* , the *Click* event is not triggered.

Example: WindowValidation Event

```

*> example of Window WindowValidationEvent <*>
proc for WindowValidation object OSPK_EVT_WINVAL_WND
(WinVal object type WindowValidationEvent)
Trace (' HpsID =', WinVal.HpsId)
Trace (' Source HpsID =', WinErr.Source.HpsId)
Trace (' ShowMessage =', WinErr.ShowMessage)
if WinVal.Accept() = TRUE
Trace(' Value Acceptable')
else
Trace('Value not Acceptable')
endproc

```

User-Interface Properties

[User-Interface Properties](#) implemented by the Java user interface objects discussed in previous sections are described in this topic.

User-Interface Properties

The Java user interface properties available to ObjectSpeak objects are listed in alphabetical order in the following table. Each property is then described in greater detail in the linked subsections.

Object properties

Altered:Boolean	Font:Font	Rollback:Boolean
---------------------------------	---------------------------	----------------------------------

AutoSelect:Boolean	Foreground:Color	RowHeight:Integer
BackBuffer:Integer	Format:Format	Selected:Boolean
Background:Color	HeaderLineCount:Integer	SelectedPageIndex:Integer
CallingRule:Rule	HpsID:String	SelectionMode:Integer
CheckMandatoryFields:Boolean	IgnoreValidation:Boolean	ShortHelp:String
Column:Column	Image:String	ShowMessage:Boolean
ColumnIndex:Integer	ImmediateReturn:Boolean	Size:Dimension
Currency:Boolean	Instance:String	Source:GuiObject
DatabaseSize:Integer	Justification:Integer	SourceName:String
DisplayMask:String	Lines:Integer	Style:Integer
DisplayPicture:String	Locale:Locale	TabStop:Boolean
DomainType:Integer	Location:Point	Text:String
Editable:Boolean	LongName:String	TopVirtualRow:Integer
EditLimit:Integer	Mandatory:Boolean	Type:Integer
EditMask:String	Mnemonic:Char	TypeString:String
Empty:Boolean	MnemonicKeycode:Integer	Validation:Boolean
Enabled:Boolean	OutputView:View	VirtualIndex:Integer
Error:Boolean	PhysicalIndex:Integer	Visible:Boolean
Focus:Boolean	PopupMenu:PopupMenu	
FocusedGuiObject:GuiObject	Response:Integer	

Altered:Boolean

For objects that have data links, this indicates whether the data in the field has been altered since the window was first displayed or since the last call to the window's *clearAltered()* method.

This property is automatically set to *True* when the data associated with any GUI object on the window is modified by the user. The application can query the Altered property to determine whether data needs to be saved. Use the *ClearAltered* method to set the value to *False* .

For example, consider a window that has a Save button. When the user clicks **Save** , the Click event procedure can query the Altered property to determine whether it actually needs to save the data. If it does save the data, the application should then call the *ClearAltered()* method and indicate that there are no more changes that need to be saved.

User-interface objects associated with data, other than windows, have their own Altered property that indicates whether the data has been modified. The Altered property is automatically set to *True* when the user modifies the data. Moreover, for every object other than the window, the Altered property is not read-only, you can both query and assign a value to the property.

A GuiObject is altered only if the data associated with that object is changed. Typing something in an edit field does not alter the edit field until you tab out of the field and the entered data is validated.

AutoSelect:Boolean

For an edit field, multiline edit, password field, and combo box, this property specifies whether text is automatically selected when the field receives focus. The default is *True*, except for multiline edit.

When an editable object gains focus, the text in the edit area is automatically selected if this property is set to *True* . The default value is *True* .

When a list box is first shown and no items are currently selected, if this property is *True* , the first item is selected when receiving focus. Also, if the list box is already shown and no items are currently selected, and this property is changed from *False* to *True* , the first item is selected when receiving focus.

Table functionality is similar to the list box functionality except that the entire row is selected only if the table's *RowSelect* property is *True* ; otherwise, only the first (non-numbering) cell of the first row is selected. When a Table is first shown, if *AutoSelect* is *True* and nothing is currently selected, the first row is selected (if *RowSelect* is *True*) or the first cell in the first row is selected (if *RowSelect* is *False*). Also, when the Table is already shown and no row or cell is currently selected, if *AutoSelect* is changed from *False* to *True* , the first one is selected when receiving focus.

For Tables (also called multicolumn list boxes or MCLBs), when `AutoSelect` is `True` :

- You *cannot* deselect the "select all" of the rows using the keyboard or mouse (but it is possible to deselect it by calling `clearSelection`).
- The selection moves with focus.
- The `setVirtualListBoxSize` and `dataRequired` events preserve any previous selection if it is less than the virtual size. For example, selecting row 1 and then calling `setVirtualListboxSize` with 50, 100 preserves the selected row even if that is not in the current virtual limits.

BackBuffer:Integer

This property represents the number of records back from the first visible record to fetch, when smooth scrolling. The value should be greater than 0 and less than $(ScrollableOccurs - VisibleOccurs)$. The ideal value should be $(ScrollableOccurs - VisibleOccurs) / 2$.

Background:Color

This property specifies the background color for the object. If the foreground or background colors are not specified, the table's color is used for the column. Group boxes have no background color; the background is transparent and displays the background color of the window.

Some objects, such as labels and check boxes, always use the background color of the window or panel. For menu items, this property and method have no effect. The `Color` object is used to specify the foreground and background colors of these objects.

CallingRule:Rule

This property specifies the parent rule for a rule, allowing to navigate back through the calling tree.

CheckMandatoryFields:Boolean

This property specifies whether push buttons and menu items verify that all mandatory fields contain data before triggering their associated action. This property is provided for backwards compatibility with previous versions of the product. We recommend you use the [Validation:Boolean](#) property for new development.

Column:Column

For certain events that occur in tables, this property provides an object reference to the table column in which the event originated. This is a read-only property.

This applies to [CellFocusGained](#) and [CellFocusLost](#) events, and to [Click](#) and [DoubleClick](#) events when they originate in a table.

ColumnIndex:Integer

For certain events that occur in tables, this property indicates the index of the table column in which the event originated. The left-most column is 1, the one to its right is 2, and so on. The numbering column (if present) is not included in the numbering. Therefore, if there is a numbering column, the column to its immediate right is 1. This is a read-only property.

This applies to [CellFocusGained](#) and [CellFocusLost](#) events, and to [Click](#) and [DoubleClick](#) events when they originate in a table.

Currency:Boolean

This property specifies whether the currency symbol should be displayed when displaying decimal data. The currency symbol can be displayed in edit fields, table cells, list boxes, and combo boxes, which are data-linked to a decimal.



Only for decimal fields.

DatabaseSize:Integer

This property specifies the size of the database.

DisplayMask:String

This property provides a mask used to format data for display when the focus is not on the field. `DisplayMask` is a property of the `Format` object. It applies to edit fields, combo boxes, list boxes, and table cells.



`DisplayMask` and `DisplayPicture` are valid for `Decimal` and `Integer` formats and *cannot* be used for `String` Formats.

DisplayPicture:String

This property is the same as [DisplayMask:String](#). It is provided for backwards compatibility with previous versions of the product.

DomainType:Integer

This property is read-only and returns the type of domain used by the combo box. The values can be one of Constants.SETDOMAIN or Constants.VIEWDOMAIN.

Editable:Boolean

This property specifies whether the data displayed in an edit field, multiline edit field, password field, or table column can be edited. To prevent the text from being edited by the user, but still allow the field to receive focus (perhaps so that text can be copied to the clipboard), set this property to *False*.

EditLimit:Integer

This property specifies the number of characters that can be entered into an edit field, multiline edit field, or password field. To limit the amount of text that can be entered, use the EditLimit property. The default value is zero (0), which specifies there is no limit on the number of characters that can be entered.

If the field is data-linked to a string or character field, the edit limit is automatically set to the length of the data-linked character field. If a field does not have a data link, it can contain a maximum of 255 characters.



For thin client applications, the EditLimit is applied only on focus lost or tab out from the input field. Unlike other AppBuilder clients, processing every key through JavaScript would slow data entry in this case.

EditMask:String

This property specifies which characters can be entered at specific locations in an editable field. EditMask is a property of the *Format* object associated with the editable field. It applies to edit fields, combo boxes, list boxes, and table cells.

For valid EditMask character values, refer to [Valid EditMask Characters](#).

Valid EditMask Characters

The valid EditMask characters that can be used in the [EditMask:String](#) property are listed in the following table:

Valid EditMask characters

Mask Character	Description	Sample Edit Mask	Sample Input	Sample Result
#	Digit placeholder	####	1234	1234
9	Digit or space placeholder	9999	1234	1234
.	Decimal separator	99.99	1234	12.34
,	Thousand separator	9,999	1234	1,234
-	Sign placeholder	-####	-1234	-1234
:	Time separator	hh:mm:ss	123001	12:30:01
/	Date separator	dd/MM/yy	121001	12/10/01

<	Converts all characters that follow to lower case	?<???	ABCD	Abcd
>	Converts all characters that follow to upper case	?>???	abcd	aBCD
&	ANSI character from 32-126 or 128-235	&&&	A2d	A2d
A or a	Alphanumeric character	AAA	A23	A23
?	Alphabet character only	???	Abc	Abc
'...'	Single quotes hold a group of literals.	'Today is' dd/MM/yy	121202	Today is 12/12/02
	Treat the next character in the mask string as a literal. Use this character to include #, &, A, or ? characters in the mask. This character is treated as a literal for masking purposes.	###\~-###\~-####	2222222222	222-222-2222
dd	Day of the month	dd/MM/yy	121202	02/12/12
EEE	Day of week	EEE dd/MM/yy	170402	Wed 17/04/02
MM	Two-digit month	MM	04	04
MMM	Three-letter month name	dd/MMM/yy	17Mar02	17/Mar/02
yy	Two-digit year	dd/MM/yy	170402	17/04/02
yyyy	Four-digit year	dd/MM/yyyy	170402	17/04/2002
mm	Minutes	hh:mm:ss	123001	12:30:01
ss	Seconds	hh:mm:ss	123001	12:30:01
hh	Hours	hh:mm:ss	123001	12:30:01
tt	AM/PM	hh:mm:ss tt	123001 AM	12:30:01 AM
<i>Literal</i>	All other symbols are displayed as literals; that is, as themselves.			



If the EditMask is set to < or >, all the characters are converted to lowercase or uppercase.

Empty: Boolean

This property indicates whether or not an editable field contains data. This is a read-only property.

Enabled: Boolean

This property specifies whether an object is enabled or disabled. Disabled objects *cannot* be edited or modified, or receive focus.

For Tables, enabling or disabling occurs on a per column basis.

For the *Timer* object, the Enabled property can be set to *False* to temporarily prevent the timer from firing. Typical applications do not need to use the Enabled property - just the *start()* and *stop()* methods.

Error:Boolean

This indicates whether the interface object contains data that is in error. This is a read-only property.

Focus:Boolean

This property has no parameters.

FocusedGuiObject:GuiObject

This property returns the guiObject that has focus.

Font:Font

This property specifies the font used to display text in an object that displays text.

For a column object, if the font is not specified, the table's default font is used for the column.

This property is not supported in thin client applications.

Foreground:Color

This property specifies the font color in all objects that display text. If the foreground or background colors are not specified, the table's color is used for the column. For menu items, this property and method have no effect. The Color object is used to specify the foreground and background colors of these objects.

Format:Format

This property specifies various formatting information for editable fields, including edit fields, combo boxes, and table cells. It specifies the *Format* object associated with the object. The *Format* object contains information about how to display and edit the text in every cell in the column. The *Format* object type and the data link must agree. For objects created in Construction Workbench, the *Format* object is automatically created and assigned to the object. Format is not used by combo boxes linked to sets (that is, static combo boxes). Refer to the [Format](#) object for more information.

HeaderLineCount:Integer

This read only property returns the count of header lines for the column.

HpsID:String

This property specifies the system identifier (HPSID) of the object. Each object must have a unique system identifier (HPSID) or unpredictable behavior may occur. The system identifier must not begin with a number and must not contain special characters (!, @, #, \$, %, ^, &, *, etc.) Underscore characters (_) may be used.

The system identifier (HPSID) can be set only once and cannot be changed. Attempting to set this property more than once generates an error. Trying to set the system identifier (HPSID) to an empty string generates an error.

IgnoreValidation:Boolean

For push buttons and menu items, this property specifies that window validation is not performed before the object's [Click](#) event is triggered. This property is provided for backwards compatibility with previous versions of the product. We recommended that you use the Validation property for new development.

Image:String

This property specifies the image file name for a Bitmap object in Window Painter. For a push button object, it is the image file name for the Bitmap of the push button.

Use a forward slash (/) for the file and directory separator. This ensures that your code is portable across platforms.

ImmediateReturn:Boolean

This property specifies whether the control can generate the immediate-return [Converse](#) event. If this property is set to *True*, a [Converse](#) event is generated. For an editable-object, this occurs when the user changes the selection of an object and moves focus to another object. For non-editable objects, this occurs when the user double-clicks an object.

This property is provided for backwards compatibility and should not be used in new application development. We recommend you use the [ImmediateReturn](#) property and that you do not use the [Converse](#) event for new applications. Use the [Click](#), [DoubleClick](#), or [FocusLost](#) event to simulate [ImmediateReturn](#) functionality.

Instance:String

This property returns the instance name of a rule.

Justification:Integer

This property specifies the horizontal justification for text in edit fields, password fields, labels, list boxes, combo boxes, and tables. Valid values, as defined in the [Constants](#) class, are:

- *LEFT*
- *CENTER*
- *RIGHT*

The [Justification](#) property is used to specify whether the text should be left-justified, centered, or right-justified. By default, text is left-justified. For a label, the horizontal justification of text within the label area is specified by the [Justification](#) property, while the vertical justification is determined by the [VerticalJustification](#) property.

Lines:Integer

This property specifies whether horizontal lines, vertical lines, both, or neither are drawn between cells in a table. Valid values, as defined in the [Constants](#) class, are:

- *HORIZONTAL_LINES*
- *VERTICAL_LINES*
- *HORIZONTAL_AND_VERTICAL_LINES*
- *NO_LINES*

Locale:Locale

This property specifies both a country and a language. Locales are used in conjunction with [Format](#) objects, as well as on the [Window](#) object. Refer to the [Locale](#) object for a list of the possible values.

Location:Point

This property specifies the position of a window or user interface object, relative to the upper left corner of the window. The property is a [Point](#) object and contains both a horizontal and vertical component.

For convenience, the [SetLocation\(X, Y\)](#) method can be used to specify the position without having to create a [Point](#) object. The size and position of the object can be queried or set using the [Size:Dimension](#) and [Location](#) properties, respectively. The size and position of a table can be queried or set using the [Size](#) and [Location](#) properties.

This property is not supported in thin client applications.

LongName:String

This property specifies the long name for a rule. The long name is stored in UPPERCASE letters. When comparing values to the long name, ensure that your value is UPPERCASE.

Mandatory:Boolean

This property specifies whether or not a field is mandatory. That is, whether it must contain data before the window closes successfully. It applies to edit fields, password fields, multiline edit fields, combo boxes, and table columns. If the [GuiObject](#) represents an editable field, then the [Mandatory](#) property indicates whether or not the field *must* contain data before the window is successfully closed.

If [Mandatory](#) is set to *True*, when window-level validation occurs, and mandatory field checking is requested, window validation fails if any of the cells in the column contain no text. See the [Validation:Boolean](#) property.



This property is not supported in the thin client applications through ObjectSpeak. It is supported through client-side javascript.

Mnemonic:Char

This property specifies the mnemonic character for buttons and other objects. Any letter in the English alphabet is valid. For push buttons, when Alt and a mnemonic key are pressed together, the push button [Click](#) event is generated. For a check box, the check box is toggled. For a radio button, it triggers the radio button. For menus and menu items, it selects or clears a menu item.



Java development does *not* support NLS characters.

MnemonicKeycode:Integer

This property specifies the mnemonic keycode for buttons and other objects; it is the integer keycode associated with the [Mnemonic:Char](#) character. This is the ASCII equivalent for the key. For push buttons, when Alt and a mnemonic key are pressed together, the push button [Click](#) event is generated. For a check box, the check box is toggled. For a radio button, the radio button is selected or cleared. For menus and menu items, a menu item is selected or cleared. The following table describes the ASCII codes.

ASCII code equivalents

• *	ASCII	• *	ASCII	• *	ASCII	• *	ASCII	• *	ASCII	• *	ASCII	• *	ASCII
0	48	9	57	I	73	R	82	a	97	j	106	s	115
1	49	A	65	J	74	S	83	b	98	k	107	t	116
2	50	B	66	K	75	T	84	c	99	l	108	u	117
3	51	C	67	L	76	U	85	d	100	m	109	v	118
4	52	D	68	M	77	V	86	e	101	n	110	w	119
5	53	E	69	N	78	W	87	f	102	o	111	x	120
6	54	F	70	O	79	X	88	g	103	p	112	y	121
7	55	G	71	P	80	Y	89	h	104	q	113	z	122
8	56	H	72	Q	81	Z	90	i	105	r	114	• *	• *

OutputView:View

This property returns the output view of a rule. This view can be mapped to any other view.

PhysicalIndex:Integer

For certain events that occur in tables, this property indicates the index (or occurrence number) of the row in the occurring view to which the table is data-linked. This applies to [CellFocusGained](#) and [CellFocusLost](#) events. It also applies to [Click](#) and [DoubleClick](#) events when they originate in a table. This is a read-only property.

PopupMenu:PopupMenu

This property sets the popup menu that can be displayed for objects on a window and the window itself, typically by right-clicking. If a given user interface object does not have a popup menu but the window does, right-clicking an object causes the window's popup menu to be displayed. Therefore, if you want only one popup menu for the window and all its objects, define a popup for the window.

Response:Integer

In [FieldValidation](#), this property specifies whether the data should be accepted, rejected, or rolled back to the last known acceptable value. The default is ACCEPT. Valid values, specified in the [Constants](#) class, are:

- *ACCEPT*
- *ROLLBACK*
- *IN_ERROR*

Rollback: Boolean

In [FieldError](#), this property specifies whether the data should be rolled back to the last known acceptable value. The default value is *False*.

RowHeight: Integer

This property specifies the height of a table row.

Selected: Boolean

This property determines if a check box, radio button, check box menu item, or radio button menu item is currently selected. This property indicates whether a menu item that can display a check mark is currently checked, or whether a menu item that can display a radio button has the radio button selected. The *Checked* property is similar to the *Selected* property, but indicates information only about the checked state.

SelectedPageIndex: Integer

This property returns the index of the selected tab page in a tab control object.

SelectionMode: Integer

This property specifies how rows can be selected for list boxes, combo boxes, and tables. Valid values, as specified in the [Constants](#) class, are:

- *SINGLE_SELECTION* - only one row at a time can be selected
- *SINGLE_RANGE_SELECTION* - (or extended) a single continuous range of rows can be selected
- *MULTIPLE_RANGE_SELECTION* - multiple ranges of rows can be selected

ShortHelp: String

This property specifies the text for the tool tip that is displayed when the mouse pointer pauses briefly on the object. If it is set to an empty string, no tool tip is displayed. It is not supported in C#.

ShowMessage: Boolean

This property specifies whether an error message box should be displayed during [FieldValidation](#) and [WindowValidation](#). By default, the message box is shown if the error condition is not removed. The default value is *True*.

This applies to the following events:

- [FieldError](#)
- [FieldValidation](#)
- [WindowError](#)
- [WindowValidation](#)

Size: Dimension

This property specifies the width and height of a window, any visible user interface object, and the window itself. The property is a [Dimension](#) object that contains both a width and height. The *Size* property of visible objects is of type *Dimension*.

For convenience, the *setSize(width, height)* method can be used to specify the width and height without having to create a *Dimension* object. The size and position of the object can be queried or set with the *Size* and *Location* properties. For example, when a *GuiObject* represents a menu item, calling the *Size* property has no effect because you cannot specify the size of a menu item. Query or set the size and position of a table using the *Size* and *Location* properties.

This property is not supported in thin client applications.

Source: GuiObject

This property provides a reference to the object that triggered the event in the form of a *GuiObject*. For more information, see the [GuiObject](#) topic. This is a read-only property.

SourceName: String

This property returns the long name of the object associated with the event.

Style:Integer

This property changes a menu item into a selectable menu item or radio button menu item. Valid values, as specified in the [Constants](#) class, are:

- PLAIN
- CHECKBOX
- RADIOBUTTON



The style of a menu item can be changed only once in Java.

TabStop:Boolean

This property specifies whether or not you can transfer keyboard focus to a user interface object by pressing the **Tab** key.

Normally, you can use the **Tab** key to move focus onto an object, but setting TabStop to *False* prevents this. Even if TabStop is False, focus can still be set on the table with the mouse (or programmatically using the *setFocus()* method) unless the field is disabled.

Text:String

This property specifies the text for labels, edit fields, multiline edit fields, editable combo boxes, and table cells. It specifies the label for group boxes and table columns and the title for group boxes. Use this property to query or set the text in the edit area. If the edit area does not have a data link, the Text property is the only way to access the text. If it has a data link, the text in the edit area can be accessed either through the Text property, or by using rules code to access the data-linked field.

TopVirtualRow:Integer

When the [DataRequired](#) event is triggered by a table that requires additional data mapped into its data-linked view, this property specifies the virtual row that appears at the top of the displayed table.

Type:Integer

This read-only property returns the type of the derived object. The type constants are defined in the Constants class. For example, if the guiObject is an editfield, the Type property returns `Constants.EDITFIELD`. Other examples include:

```
Constants.DATE_FORMAT
Constants.DECIMAL_FORMAT
Constants.LONGINT_FORMAT
Constants.SHORTINT_FORMAT
Constants.STRING_FORMAT
Constants.TIME_FORMAT
```

TypeString:String

This specifies the type of event in a [DataRequired](#) event. The string is one of the following:

- *ListBoxTop*
- *ListBoxBottom*
- *OutOfRange*

Validation:Boolean



For push buttons and menu items, this specifies whether window validation must be successfully performed before the object's [Click](#) event is triggered. Setting Validation to *True* is equivalent to setting [IgnoreValidation:Boolean](#) to *False* and [CheckMandatoryFields:Boolean](#) to *True*.

To cause [Window-level Validation](#) to occur when an object is clicked, set its Validation property to *True*. Window-level validation allows the application to verify that the user has specified all required information, and that the information in the various fields is acceptable.

For backwards compatibility, push buttons and menu items contain not only the Validation property but also two Boolean properties named `IgnoreValidation` and `CheckMandatoryFields`. The Validation property and the `IgnoreValidation` and `CheckMandatoryFields` properties represent two slightly different approaches to validation. If the Validation property is used to enable window validation, the check on mandatory fields is always performed. However, if `IgnoreValidation` is used to enable validation, then the check on mandatory fields is done *only* if `CheckMandatoryFields` is *True*. We recommend you use the `Validate` property for new applications because checking mandatory fields should be routinely performed as part of window validation.

Properties / Objects	C h e c k B o x	C o l u m n	C o m b o B o x	E d i t F i e l d	E l l i p s e	G r o u p B o x	L a b e l	L i s t B o x	M e n u	M e n u B a r	M e n u I t e m	M u l t i L i n e E d i t	P a s s w o r d F i e l d	P u s h B u t t o n	R a d i o B u t t o n	R e c t a n g l e	T a b l e	F o r m a t	G u i O b j e c t	R u l e	W i n d o w	M e s s a g e B o x	P o p M e n u	T i m e r	
Accelerator									J																
Altered	J	J	J	J	J		J					J	J		J	J	J		J		J				
Argument1																							J		
Argument2																							J		
Argument3																							J		
AutoSelect			J	J																					
AutoTab				J																					
Background	HJ	HJ	HJ	HJ	J		HJ	HJ				HJ	HJ	HJ	HJ	J	HJ		HJ		HJ				
Border: Boolean				J									J												
BorderStyle																									
ButtonType																							J		
CallingRule																					J				
Checked											HJ														
Check Mandatory Fields											J		J												
Column																	J								
Column Index																									
Country																									
Currency																									
DataLink	HJ		HJ	HJ								HJ	HJ		HJ									J	
Delay																									
Display Mask																									
Display Picture																									
Editable		J	J	HJ								J	HJ												
EditLimit		J	J	J								J	J												
EditMask																			J						

appbuilder.util. AbfBlob	implements java.sql.Clob, java.sql.Blob
	public void map(Reader from) throws AbfDataException
	public void map(InputStream from) throws AbfDataException
	public void map(byte[] from) throws AbfDataException
	public void map(char[] from) throws AbfDataException
	public void map(String filename)
	public char[] getChars() throws IOException
	public byte[] getBytes() throws IOException
	public String getFilename()
appbuilder.util. AbfBoolean	public void map(Boolean value)
	public boolean getJavaValue()
appbuilder.util. AbfDecimal	public void map(BigDecimal d)
	public BigDecimal getJavaValue()
appbuilder.util. AbfDouble	public void map(double value)
	public double getJavaValue()
appbuilder.util. AbfFloat	public void map(double value)
	public void map(float value)
	public double getJavaValue()
appbuilder.util. AbfLongInt	public void map(long value)
	public long getJavaValue()
appbuilder.util. AbfInt	public void map(int value)
	public int getJavaValue()
appbuilder.util. AbfTime	public void map(Date date)
	public Date getJavaValue()
appbuilder.util. AbfTimeStamp	public void map(Date date)
	public Date getJavaValue()
appbuilder.util. AbfShortInt	public void map(short value)
	public void map(int value)
	public short getJavaValue()
appbuilder.util. AbfString	public void map(String value)
	public String getJavaValue()
appbuilder.util. AbfDataChangeListener	public void dataChange(AbfDataChangeEvent evt)
	public void preDataChange(AbfDataChangeEvent evt)
	public void sizeChange(AbfDataChangeEvent evt)
appbuilder.util. AbfDataChangeEvent	public AbfDataObject getDataObject()
appbuilder.util. AbfSystem	void init(String appbuilderIniUrl)
	public void appTrace(int level, String s)
	public static String deriveClassName(int type, String longName) a
	public static String deriveObjectName(int type, String longName) b

appbuilder. AbfModule  This is a base class of rule types.	public static AbfStruct start(String ruleClassName, HpsView iView, AbfStruct oView)
	public AbfModule getCallingRule()
	public AbfStruct run (AbfStruct iView)
appbuilder. AbfClientModule  This is a base class for servlet and GUI rules.	public void post(HpsPostEvent e)
appbuilder. AbfPostEvent	public AbfPostEvent (Object source, String sourceName, String subject, HpsView view)
	public AbfStruct getView()
	public String getSourceName ()
	public String getSubject()
	public Object getSourceObject()
	public String getParam()

a. Given an object type and it's long name, it returns the generated java class name. The valid types are `RULE_TYPE`, `VIEW_TYPE`, `VIEWARRAY_TYPE`, `SET_TYPE` and `COMPONENT_TYPE` as defined in `AbfSystem`.


b. Given an object type and it's long name, it returns the generated name of the instance variable. In addition to the types in `deriveClassName`, `AbfSystem.WINDOW_TYPE`, `FIELD_TYPE`, `OBJECT_TYPE` can be used.

Supported Methods in CSharp

Supported Methods in C#

This appendix presents a full list of supported methods in C# ObjectSpeak.

All thin client features of Java are not supported in C# since thin client is not supported.

 If an object inherits any class, then all methods and properties of the inherited class are available in the object.

Accelerator

C# support for Accelerator
Constructors
CONSTRUCTOR(ODE_eCHAR, ODE_eLONG)
CONSTRUCTOR(ODE_eLONG, ODE_eLONG)
Properties
ALT
CTRL

KeyChar
KeyCode
Modifiers
SHIFT
VK_F1
VK_F2
VK_F3
VK_F4
VK_F5
VK_F6
VK_F7
VK_F8
VK_F9
VK_F10
VK_F11
VK_F12

ActivateEvent

C# support for ActivateEvent
Properties
HpsId

Array

C# support for Array
Methods
ELEM(ODE_eINDEX, ODE_eELEM)
ELEM(ODE_eINDEX)
INSERT(ODE_eINDEX)
REPLACE(ODE_eINDEX, ODE_eELEM)
RESIZE(ODE_eELEM)
SIZE()

CellFocusGainedEvent

C# support for CellFocusGainedEvent
Properties
COLUMN
COLUMNINDEX

HPSID
PHYSICALINDEX
SOURCE
VIRTUALINDEX

CellFocusLostEvent

C# support for CellFocusLostEvent
Properties
COLUMN
COLUMNINDEX
HPSID
PHYSICALINDEX
SOURCE
VIRTUALINDEX

CheckBox

C# support for CheckBox
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eString)
Events
CLICK(ODE_eOBJECT)
DOUBLECLICK(ODE_eOBJECT)
FOCUSGAINED(ODE_eOBJECT)
FOCUSLOST(ODE_eOBJECT)
FIELDERROR(ODE_eOBJECT)
Properties
ALTERED - has Set method
AUTOSELECT
DATALINK - has Set method
EDITABLE - has Set method
EMPTY
ERROR
IMMEDIATERETURN - has Set method
MANDATORY - has Set method

MNEMONIC - has Set method
MNEMONICKEYCODE - has Set method
POPUPMENU - has Get/Set methods
SELECTED - has Set method
TABSTOP - has Set method
TEXT - has Set method

ClickEvent

C# support for ClickEvent
Properties
COLUMN
COLUMNINDEX
HPSID
PHYSICALINDEX
SOURCE
VIRTUALINDEX

CloseEvent

C# support for CloseEvent
Properties
HPSID

Color

C# support for Color
Constructors
CONSTRUCTOR(ODE_eLONG, ODE_eLONG, ODE_eLONG)
CONSTRUCTOR(ODE_eOBJECT)
Properties
BLACK
BLUE
BROWN
CYAN
DARKBLUE
DARKCYAN
DARKGRAY
DARKGREEN

DARKMAGENTA
DARKRED
DARKYELLOW
GRAY
GREEN
LIGHTGRAY
MAGENTA
PINK
RED
TURQUOISE
WHITE
YELLOW

Column

C# support for Column
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
FIELDERROR(ODE_eOBJECT)
Properties
ALTERED - has Set method
EDITABLE - has Set method
EDITLIMIT - has Set method
EMPTY
ERROR
FIELDPATH - has Get/Set methods
FORMAT - has Set method
HEADERLINECOUNT - has Get method
IMMEDIATERETURN - has Set method
JUSTIFICATION - has Get/Set methods
MANDATORY - has Set method
POPUPMENU - has Get/Set methods
SETLINK - has Get/Set methods
WIDTH - has Set methods
Methods

ADDHEADER(ODE_eSTRING)
ADDHEADER(ODE_eSTRING, ODE_eLONG)
GETHEADER()
GETHEADER(ODE_eLONG)
GETSCALEDWIDTH(ODE_eLONG)
REMOVEHEADER(ODE_eSTRING)
ODE_eVOID SETHEADER(ODE_eString, ODE_eInt)
SETSCALEDWIDTH

ComboBox

C# support for ComboBox
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
CLICK(ODE_eOBJECT)
DOUBLECLICK(ODE_eOBJECT)
FIELDERROR(ODE_eOBJECT)
FIELDVALIDATION(ODE_eOBJECT)
FOCUSGAINED(ODE_eOBJECT)
FOCUSLOST(ODE_eOBJECT)
SELECT(ODE_eOBJECT)
Properties
ALTERED - has Set method
AUTOSELECT - has Set method
DATALINK - has Set method
DOMAINTYPE - has Get method
EDITABLE - has Set method
EDITLIMIT - has Set method
EMPTY
ERROR
FIELDPATH - has Get/Set methods
FORMAT - has Set method
IMMEDIATEReturn - has Set method
ISAUTOSELECT
JUSTIFICATION - has Get/Set methods

MANDATORY - has Set method
POPUPMENU - has Get/Set methods
SETLINK - has Set method
SELECTEDINDEX - has Set method
TABSTOP - has Set method
TEXT - has Set method
VIEWLINK - has Set method
Methods
GETLISTLINK
SETLISTLINK(ODE_eOBJECT)

CommErrorEvent

C# support for CommErrorEvent
Properties
HPSID

Constants

C# support for Constants
Properties
ACCEPT
ALL_FIRST_UPPER_CASE
ALT
ASYNC_EVENT
BITMAP
BOOLEAN_FORMAT
CANCEL
CENTER
CENTER_JUSTIFY
CHECKBOX
CHECKBOX_MENUITEM
COLUMN
COMBOBOX
COORDINATE_CHAR
COORDINATE_PIXEL
CTRL
DATE_FORMAT
DECIMAL_FORMAT

DEFAULT_BUTTONS
DEFAULT_CASE
DOUBLE_FORMAT
EDITFIELD
ELLIPSE
ERROR
FILEEDITOR
FIRST_UPPER_CASE
FLOAT_FORMAT
GROUPBOX
HOTSPOT
HORIZONTAL_AND_VERTICAL_LINES
HORIZONTAL_LINES
INFORMATION
IN_ERROR
INT_FORMAT
INTERFACE_EVENT
LABEL
LANDP_EVENT
LANDP_REQUEST_EVENT
LANDP_SYSTEM_EVENT
LEFT
LEFT_JUSTIFY
LISTBOX
LONGINT_FORMAT
LOWER_CASE
MENU
MENUITEM
MULTILINEEDIT
MULTIPLE_RANGE_SELECTION
NO
NO_LINES
OK
OK_BUTTON
OK_CANCEL
PANE
PASSWORDFIELD
PLAIN
PLAIN_MENUITEM

POPUPMENU
PROGRESSBAR
PUSHBUTTON
QUESTION
RADIOBUTTON
RADIOBUTTON_MENUITEM
RECTANGLE
RIGHT
RIGHT_JUSTIFY
ROLLBACK
SHIFT
SHORTINT_FORMAT
SINGLE_RANGE_SELECTION
SINGLE_SELECTION
STRING_FORMAT
SYSTEM_EVENT
TABBEDPANE
TABLE
TIME_FORMAT
UPPER_CASE
USER_EVENT
WARNING
WINDOW
YES
YES_NO
YES_NO_CANCEL
VERTICAL_LINES
VK_DELETE
VK_F1
VK_F2
VK_F3
VK_F4
VK_F5
VK_F6
VK_F7
VK_F8
VK_F9
VK_F10
VK_F11

VK_F12

VK_INSERT

ConverseEvent

C# support for ConverseEvent

Properties

HPSID

DataRequiredEvent

C# support for DateRequiredEvent

Properties

DATABASESIZE

HPSID

REFRESH

SETDATABASESIZE

SETREFRESH

SETTOPVIRTUALROW

SOURCE

TOPVIRTUALROW

TYPESTRING

DecimalFormat

C# support for DecimalFormat

Properties

CURRENCY - has Set method

DISPLAYMASK - has Set method

DISPLAYPICTURE - has Set method

EDITMASK - has Set method

MAXIMUMSET

MINIMUMSET

Methods

DECIMALTOSTRING(ODE_eOBJECT, ODE_eSTRING, ODE_eOBJECT)
--

DISPLAYSTRING(ODE_eOBJECT)

EDITSTRING(ODE_eOBJECT)

SETMAXIMUM(ODE_eOBJECT)

SETMINIMUM(ODE_eOBJECT)

STRINGTODECIMAL(ODE_eSTRING, ODE_eSTRING, ODE_eOBJECT)

Dimension

C# support for Dimension
Constructors
CONSTRUCTOR(ODE_eLONG, ODE_eLONG)
Properties
HEIGHT - has Set method
WIDTH - has Set method

DoubleClickEvent

C# support for DoubleClickEvent
Properties
COLUMN
COLUMNINDEX
HPSID
PHYSICALINDEX
SOURCE
VIRTUALINDEX

DoubleFormat

C# support for DoubleFormat
Properties
CURRENCY - has Set method
DISPLAYMASK - has Set method
DISPLAYPICTURE - has Set method
EDITMASK - has Set method
MAXIMUMSET
MINIMUMSET
Methods
DISPLAYSTRING(ODE_eOBJECT)
DOUBLETOSTRING(ODE_eOBJECT, ODE_eSTRING, ODE_eOBJECT)
EDITSTRING(ODE_eOBJECT)
SETMAXIMUM(ODE_eOBJECT)
SETMINIMUM(ODE_eOBJECT)
STRINGTODOUBLE(ODE_eSTRING, ODE_eSTRING, ODE_eOBJECT)

EditField

C# support for EditField
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
CLICK(ODE_eOBJECT)
DOUBLECLICK(ODE_eOBJECT)
FIELDERROR(ODE_eOBJECT)
FIELDVALIDATION(ODE_eOBJECT)
FOCUSGAINED(ODE_eOBJECT)
FOCUSLOST(ODE_eOBJECT)
Properties
ALTERED - has Set method
AUTOSELECT - has Set method
AUTOTAB - has Set method
BORDER - has Set method
DATALINK - has Set method
EDITABLE - has Set method
EDITLIMIT
EMPTY
ENABLED - has Set method
ERROR
FORMAT - has Set method
IMMEDIATERETURN - has Set method
JUSTIFICATION - has Get/Set methods
MANDATORY - has Set method
POPUPMENU - has Get/Set methods
SETLINK - has Set method
TABSTOP - has Set method
TEXT - has Set method

Ellipse

C# support for Ellipse

Inherits
GuiObject
Constructors
CONSTRUCTOR()
Events
CLICK(ODE_eOBJECT)
DOUBLECLICK(ODE_eOBJECT)

EnterKeyEvent

C# support for EnterKeyEvent
Properties
ENTERKEY
HPSID
SOURCE

FieldErrorEvent

C# support for FieldErrorEvent
Properties
COLUMNINDEX
HPSID
PHYSICALINDEX
ROLLBACK - has Set method
SHOWMESSAGE - has Set method
SOURCE
VIRTUALINDEX

FieldValidationEvent

C# support for FieldValidationEvent
Properties
COLUMNINDEX
HPSID
PHYSICALINDEX
RESPONSE
SETRESPONSE
SHOWMESSAGE - has Set method
SOURCE

FileEditor

C# support for FileEditor
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
CLICK(ODE_eOBJECT)
DOUBLECLICK(ODE_eOBJECT)
FIELDERROR(ODE_eOBJECT)
FIELDVALIDATION(ODE_eOBJECT, ODE_eOBJECT)
FIELDVALIDATION(ODE_eOBJECT)
FOCUSGAINED(ODE_eOBJECT)
FOCUSLOST(ODE_eOBJECT)
Properties
ALTERED - has Set method
AUTOSELECT - has Set method
DATALINK - has Set method
EDITABLE - has Set method
EDITLIMIT - has Set method
EMPTY
ERROR
IMMEDIATEReturn - has Set method
INSERTBREAK - has Set method
INSERTTAB - has Set method
MANDATORY - has Set method
POPUPMENU - has Get/Set methods
TABSTOP - has Set method
TEXT - has Set method
WORDWRAP - has Set method

FloatFormat

C# support for FloatFormat
Properties

CURRENCY - has Set method
DISPLAYMASK - has Set method
DISPLAYPICTURE - has Set method
EDITMASK - has Set method
MAXIMUMSET
MINIMUMSET
Methods
DISPLAYSTRING(ODE_eOBJECT)
EDITSTRING(ODE_eOBJECT)
FLOATTOSTRING(ODE_eOBJECT, ODE_eSTRING, ODE_eOBJECT)
SETMAXIMUM(ODE_eOBJECT)
SETMINIMUM(ODE_eOBJECT)
STRINGTOFLOAT(ODE_eSTRING, ODE_eSTRING, ODE_eOBJECT)

FocusGainedEvent

C# support for FocusGainedEvent
Properties
HPSID
SOURCE

FocusLostEvent

C# support for FocusLostEvent
Properties
HPSID
SOURCE

Font

C# support for Font
Constructors
CONSTRUCTOR(ODE_eOBJECT)
CONSTRUCTOR(ODE_eSTRING, ODE_eLONG, ODE_eFLOAT)
Properties
BOLD
DISPLAYNAME
FONT
FONTNAMES

ITALIC
MODERN10
MODERN12
MODERN8
PLAIN
ROMAN10
ROMAN12
ROMAN14
ROMAN18
ROMAN24
ROMAN8
SIZE
STYLE
SWISS10
SWISS12
SWISS14
SWISS18
SWISS24
SWISS8
SYSTEMFONT8
Methods
GETFONT(ODE_eOBJECT)
GETFONTNAMES(ODE_eOBJECT)
GETSIZE(ODE_eFLOAT)
GETSTYLE(ODE_eLONG)

Format

C# support for Format
Properties
DISPLAYMASK - has Set method
DISPLAYPICTURE - has Set method
EDITMASK - has Set method
Methods
DISPLAYSTRING(ODE_eOBJECT)
EDITSTRING(ODE_eOBJECT)

GroupBox

C# support for GroupBox
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Properties
TEXT - has Set method

GuiObject

C# support for GuiObject
Properties
BACKGROUND - has Set method
ENABLED - has Is/Set methods
FOCUS - has Has method
FONT - has Get/Set methods
FOREGROUND - has Set method
HPSID - has Set method
LOCATION - has Set method
SHORTHELP - has Set method
SIZE - has Set method
TYPE
VISIBLE - has Is/Set methods
Methods
SETFOCUS()
SETSIZE(ODE_eLONG, ODE_eLONG)
SETLOCATION(ODE_eLONG, ODE_eLONG)

HeaderClickEvent

C# support for HeaderClickEvent
Properties
COLUMN
COLUMNINDEX
HPSID
SORTINGORDER - has Set method
SOURCE

USEDEFAULTSORTING - has Set method

InitializeEvent

C# support for InitializeEvent

Properties

HPSID

IntFormat

C# support for IntFormat

Properties

CURRENCY - has Set method

DISPLAYMASK - has Set method

DISPLAYPICTURE - has Set method

EDITMASK - has Set method

MAXIMUMSET

MINIMUMSET

Methods

DISPLAYSTRING(ODE_eOBJECT)

EDITSTRING(ODE_eOBJECT)

INTTOSTRING(ODE_eOBJECT, ODE_eSTRING, ODE_eOBJECT)

SETMAXIMUM(ODE_eOBJECT)

SETMINIMUM(ODE_eOBJECT)

STRINGTOINT(ODE_eSTRING, ODE_eSTRING, ODE_eOBJECT)

IWindow

C# support for IWindow

Events

CLOSE(ODE_eOBJECT)

CONVERSE(ODE_eOBJECT)

ENTERKEY(ODE_eOBJECT)

INITIALIZE(ODE_eOBJECT)

TERMINATE(ODE_eOBJECT)

WINDOWERROR(ODE_eOBJECT)

WINDOWVALIDATION(ODE_eOBJECT)

Properties

ALTERED - has Is/Set methods

CLIENTSIZE
CLIENTSIZESET - has Is method
DEFAULTBUTTON
FINDFOCUSEDGUIOBJECT
FOCUSEDGUIOBJECT
FOCUSOWNER - has Get method
HELPTOPIC - has Get method
MAXIMIZABLE - has Is/Set methods
MENUBAR
MINIMIZABLE - has Is/Set methods
POPUPMENU - has Get/Set methods
RESIZABLE - has Is/Set methods
TEXT - has Set method
TITLE - has Set method
Methods
ADDCHILD(ODE_eOBJECT)
CLEARALTERED()
CLEARSELECTION()
CLEARWINDOWCHANGES()
GETABFMENUBAR()
GETDEFAULTPUSHBUTTON()
GETSCALEDCLIENTSIZE()
PRINTFRAME()
SETABFMENUBAR(ODE_eOBJECT)
SETDEFAULTPUSHBUTTON(ODE_eOBJECT)
SETSCALEDCLIENTSIZE(ODE_eOBJECT)
SHOWMESSAGEBOX(ODE_eSTRING, ODE_eSTRING, ODE_eLONG, ODE_eLONG)
SHOWMESSAGEBOX(ODE_eSTRING, ODE_eLONG)
TERMINATE()
UPDATEDISPLAY()

Label

C# support for Label
GuiObject
Constructors
CONSTRUCTOR()
Events
CLICK(ODE_eOBJECT)

DOUBLECLICK(ODE_eOBJECT)
Properties
HORIZONTALTEXTPOSITION - has Set method
IMAGE - has Set method
JUSTIFICATION - has Get/Set methods
MNEMONIC - has Set method
MNEMONICKEYCODE - has Set method
POPUPMENU - has Get/Set methods
TEXT - has Set method
VERTICALJUSTIFICATION - has Get/Set method
Methods
SETAUTOSIZE(ODE_eBOOL)
SETVERTICALTEXTPOSITION(ODE_eLONG)

ListBox

C# support for ListBox
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
CLICK(ODE_eOBJECT)
DOUBLECLICK(ODE_eOBJECT)
FIELDERROR(ODE_eOBJECT)
FOCUSGAINED(ODE_eOBJECT)
FOCUSLOST(ODE_eOBJECT)
Properties
ALTERED
EDITABLE - has Set method
EMPTY
ERROR
FIELDPATH - has Get/Set methods
FORMAT - has Set method
IMMEDIATERETURN - has Set method
ISAUTOSELECT
JUSTIFICATION - has Get/Set methods
MANDATORY - has Set method

NEXTSELECTEDINDEX - has Get method
POPUPMENU - has Set method
SELECTEDINDEX - has Set method
SELECTIONMODE - has Set method
TABSTOP - has Set method
VIEWLINK - has Set method
Methods
CLEARSELECTION()
GETFIELDPATHSTRING()
GETNEXTSELECTEDINDEX(ODE_eLONG)
NEXTSELECTEDINDEX(ODE_eLONG)
RESETSELECTEDINDEX(ODE_eLONG)
RESETSELECTIONINTERVAL(ODE_eLONG, ODE_eLONG)
SETAUTOSELECT(ODE_eBOOL)
SETFIELDPATHSTRING(ODE_eSTRING)
SETSELECTIONINTERVAL(ODE_eLONG, ODE_eLONG)

LongIntFormat

C# support for LongIntFormat
Properties
CURRENCY - has Set method
DISPLAYMASK - has Set method
DISPLAYPICTURE - has Set method
EDITMASK - has Set method
MAXIMUMSET
MINIMUMSET
Methods
DISPLAYSTRING(ODE_eOBJECT)
EDITSTRING(ODE_eOBJECT)
LONGINTTOSTRING(ODE_eOBJECT, ODE_eSTRING, ODE_eOBJECT)
SETMAXIMUM(ODE_eOBJECT)
SETMINIMUM(ODE_eOBJECT)
STRINGTOLONGINT(ODE_eSTRING, ODE_eSTRING, ODE_eOBJECT)

Menu

C# support for Menu
Inherits

GuiObject
Constructors
CONSTRUCTOR(ODE_eSTRING)
CONSTRUCTOR()
Events
CLICK(ODE_eOBJECT)
Properties
ACCELERATOR - has Set method
CHECKED - has Is/Set methods
CHECKMANDATORYFIELDS - has Set method
COUNT
IGNOREVALIDATION - has Set method
MNEMONIC - has Set method
MNEMONICKEYCODE - has Set method
SELECTED - has Is/Set methods
STYLE - has Set method
TEXT - has Get/Set methods
VALIDATION - has Set method
Methods
ADD(ODE_eOBJECT)
ADDSEPARATOR()
GETITEM(ODE_eLONG)
GETITEMCOUNT

MenuBar

C# support for MenuBar
Inherits
GuiObject
Constructors
CONSTRUCTOR()
Properties
MENUCOUNT - has Get method
Methods
ADD(ODE_eOBJECT)
GETMENU(ODE_eLONG)

MenuItem

C# support for MenuItem
Inherits
GuiObject
Constructors
CONSTRUCTOR(ODE_eSTRING)
CONSTRUCTOR()
Events
CLICK(ODE_eOBJECT)
Properties
ACCELERATOR - has Set method
CHECKED - has Is/Set methods
CHECKMANDATORYFIELDS - has Set method
COUNT
IGNOREVALIDATION - has Set method
MNEMONIC - has Set method
MNEMONICKEYCODE - has Set method
SELECTED - has Is/Set methods
STYLE - has Set method
TEXT - has Get/Set methods
VALIDATION - has Set method
Methods
ADD(ODE_eOBJECT)
ADDSEPARATOR()
GETITEM(ODE_eLONG)
GETITEMCOUNT

MessageBox

C# support for MessageBox
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eOBJECT, ODE_eSTRING, ODE_eSTRING, ODE_eLONG, ODE_eLONG)
Properties
ARGUMENT1 - has Set method
ARGUMENT2 - has Set method
ARGUMENT3 - has Set method
BUTTONTYPE - has Set method
LOCALE

MESSAGE - has Set method
MESSAGETYPE - has Set method
PARENT
TITLE - has Set method
Methods
SETABFLOCALE(ODE_eOBJECT)
SETABFPARENT(ODE_eOBJECT)
SHOW()

MultiLineEdit

C# support for MultiLineEdit
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
CLICK(ODE_eOBJECT)
DOUBLECLICK(ODE_eOBJECT)
FIELDERROR(ODE_eOBJECT)
FIELDVALIDATION(ODE_eOBJECT, ODE_eOBJECT)
FIELDVALIDATION(ODE_eOBJECT)
FOCUSGAINED(ODE_eOBJECT)
FOCUSLOST(ODE_eOBJECT)
Properties
ALTERED - has Set method
AUTOSELECT - has Set method
DATALINK - has Set method
EDITABLE - has Set method
EDITLIMIT - has Set method
EMPTY
ERROR
IMMEDIATEReturn - has Set method
INSERTBREAK - has Set method
INSERTTAB - has Set method
MANDATORY - has Set method
POPUPMENU - has Set method
TABSTOP - has Get/Set methods

TEXT - has Set method

NodeEvent

C# support for NodeEvent

Properties

HPSID

TREENODE

SOURCE

PageSelectEvent

C# support for PageSelectEvent

Properties

HPSID

SELECTEDPAGEINDEX

SOURCE

Pane

C# support for Pane

Inherits

[GuiObject](#)

Constructors

CONSTRUCTOR()

Events

CLICK(ODE_eOBJECT)

DOUBLECLICK(ODE_eOBJECT)

FOCUSGAINED(ODE_eOBJECT)

FOCUSLOST(ODE_eOBJECT)

Properties

BORDERSTYLE - has Set method

LOWERED_BEVEL

NO_BORDER

OPAQUE - has Set method

POPUPMENU - has Set method

RAISED_BEVEL

TABSTOP - has Set method

Methods

ADD(ODE_eOBJECT)

PasswordField

C# support for PasswordField
Inherits
EditField
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
CLICK(ODE_eOBJECT)
DOUBLECLICK(ODE_eOBJECT)
FIELDERROR(ODE_eOBJECT)
FIELDVALIDATION(ODE_eOBJECT)
FOCUSGAINED(ODE_eOBJECT)
FOCUSLOST(ODE_eOBJECT)
Properties
ALTERED - has Set method
AUTOSELECT - has Set method
AUTOTAB - has Set method
BORDER - has Set method
DATALINK - has Set method
EDITABLE - has Set method
EDITLIMIT - has Set method
EMPTY
ENABLED - has Set method
ENABLED - has Set method
ERROR
FORMAT - has Set method
IMMEDIATEReturn - has Set method
JUSTIFICATION - has Get/Set methods
MANDATORY - has Set method
POPUPMENU - has Get/Set methods
SETLINK - has Set method
TABSTOP - has Set method
TEXT - has Set method

Point

C# support for Point
Constructors
CONSTRUCTOR(ODE_eLONG, ODE_eLONG)
Properties
X - has Set method
Y - has Set method

PopupMenu

C# support for PopupMenu
Inherits
GuiObject
Constructors
CONSTRUCTOR()
Methods
ADD(ODE_eOBJECT)
ADDSEPARATOR()

PushButton

C# support for PushButton
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
CLICK(ODE_eOBJECT)
FOCUSGAINED(ODE_eOBJECT)
FOCUSLOST(ODE_eOBJECT)
Properties
CHECKMANDATORYFIELDS - has Set method
HORIZONTALTEXTPOSITION - has Set method
IGNOREVALIDATION - has Set method
IMAGE - has Set method
MNEMONIC - has Set method
MNEMONICKEYCODE - has Set method

POPUPMENU - has Set method
PRESSED
SETTABSTOP
TEXT - has Set method
VALIDATION - has Set method
VERTICALTEXTPOSITION
Methods
SETVERTICALTEXTTOIMAGEPOSITION(ODE_eLONG)

RadioButton

C# support for RadioButton
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
CLICK(ODE_eOBJECT)
DOUBLECLICK(ODE_eOBJECT)
FIELDERROR(ODE_eOBJECT)
FOCUSGAINED(ODE_eOBJECT)
FOCUSLOST(ODE_eOBJECT)
Properties
ALTERED - has Set method
DATALINK - has Set method
EDITABLE - has Set method
EMPTY
ERROR
IMMEDIATEReturn - has Set method
MANDATORY - has Set method
MNEMONIC - has Set method
MNEMONICKEYCODE - has Set method
POPUPMENU - has Set method
SELECTED - has Set method
TABSTOP - has Set method
TEXT - has Set method

Rectangle

C# support for Rectangle
Inherits
GuiObject
Constructors
CONSTRUCTOR()
Events
CLICK(ODE_eOBJECT)
DOUBLECLICK(ODE_eOBJECT)

RowFocusGainedEvent

C# support for RowFocusGainedEvent
Properties
HPSID
PHYSICALINDEX
SOURCE
VIRTUALINDEX

RowFocusLostEvent

C# support for RowFocusLostEvent
Properties
HPSID
PHYSICALINDEX
SOURCE
VIRTUALINDEX

Rule

C# support for Rule
Events
ACTIVATE(ODE_eOBJECT)
CHILDRULEEND(ODE_eOBJECT)
COMMERROR(ODE_eOBJECT)
CONVERSE(ODE_eOBJECT)
INITIALIZE(ODE_eOBJECT)
PARENTRULEEND(ODE_eOBJECT)
POST(ODE_eOBJECT)

RULEEND(ODE_eOBJECT)
SQLERROR(ODE_eOBJECT)
TERMINATE(ODE_eOBJECT)
Properties
ACTIVEWINDOW
CALLINGRULE - has Get method
IMPNAME - has Get method
INSTANCE
LONGNAME - has Get method
SHORTNAME - has Get method
WINDOW - has Get method
Methods
GETIMPNAME()
GETINSTANCENAME()
GETTARGETWINDOW()
FINDGUIOBJECT(ODE_eSTRING)
FINDGUIOBJECT(ODE_eSTRING, ODE_eLONG)
POSTTO(ODE_eOBJECT, ODE_eSTRING)
POSTTO(ODE_eOBJECT, ODE_eSTRING, ODE_eOBJECT)
POSTTO(ODE_eOBJECT, ODE_eSTRING, ODE_eOBJECT, ODE_eSTRING)
POSTTOCHILD(ODE_eSTRING, ODE_eSTRING)
POSTTOCHILD(ODE_eSTRING, ODE_eSTRING, ODE_eOBJECT)
POSTTOCHILD(ODE_eSTRING, ODE_eSTRING, ODE_eOBJECT, ODE_eSTRING)
POSTTOPARENT(ODE_eSTRING, ODE_eOBJECT)
POSTTOPARENT(ODE_eSTRING)
POSTTOPARENT(ODE_eSTRING, ODE_eOBJECT, ODE_eSTRING)
QUERYUSERAUTHENTICATION()
SETHELPPFILE(ODE_eSTRING)
SETUSERAUTHENTICATION(ODE_eSTRING, ODE_eSTRING)
SHOWHELPTOPIC(ODE_eSTRING)
TERMINATE()
TRACE(ODE_eSTRING)

Set

C# support for Set
Methods
ADDSETITEM(ODE_eSTRING, ODE_eDECIMALRef, ODE_eLONG)
ADDSETITEM(ODE_eSTRING, ODE_eINTRef, ODE_eLONG)

ADDSETITEM(ODE_eSTRING, ODE_eSTRINGRef, ODE_eSTRING, ODE_eLONG)
ADDSETITEM(ODE_eSTRING, ODE_eDECIMALRef, ODE_eSTRING, ODE_eLONG)
ADDSETITEM(ODE_eSTRING, ODE_eINTRef, ODE_eSTRING, ODE_eLONG)
ADDSETITEM(ODE_eSTRING, ODE_eSTRINGRef)
ADDSETITEM(ODE_eOBJECT)
ADDSETITEM(ODE_eSTRING, ODE_eINTRef)
ADDSETITEM(ODE_eSTRING, ODE_eSTRINGRef, ODE_eLONG)
ADDSETITEM(ODE_eSTRING, ODE_eDECIMALRef)
GETSETITEM(ODE_eDECIMALRef)
GETSETITEM(ODE_eINTRef)
GETSETITEM(ODE_eSTRINGRef)
GETSETITEMFROMDISPLAY(ODE_eSTRING)
REFRESH()

SetItem

C# support for SetItem
Constructors
CONSTRUCTOR(ODE_eSTRING, ODE_eSTRINGRef, ODE_eLONG)
CONSTRUCTOR(ODE_eSTRING, ODE_eDECIMALRef, ODE_eSTRING, ODE_eLONG)
CONSTRUCTOR(ODE_eSTRING, ODE_eDECIMALRef, ODE_eLONG)
CONSTRUCTOR(ODE_eSTRING, ODE_eINTRef, ODE_eLONG)
CONSTRUCTOR(ODE_eSTRING, ODE_eINTRef, ODE_eSTRING, ODE_eLONG)
CONSTRUCTOR(ODE_eSTRING, ODE_eINTRef)
CONSTRUCTOR(ODE_eSTRING, ODE_eSTRINGRef)
CONSTRUCTOR(ODE_eSTRING, ODE_eSTRINGRef, ODE_eSTRING, ODE_eLONG)
CONSTRUCTOR(ODE_eSTRING, ODE_eDECIMALRef)
Properties
DISABLED
ENABLED
GETDISPLAY
GETENCODING
GETTEXT
NONSELECTABLE
STATE - has Set method

ShortIntFormat

C# support for ShortIntFormat

Properties
CURRENCY - has Set method
DISPLAYMASK - has Set method
DISPLAYPICTURE - has Set method
EDITMASK - has Set method
MAXIMUMSET
MINIMUMSET
Methods
DISPLAYSTRING(ODE_eOBJECT)
EDITSTRING(ODE_eOBJECT)
SETMAXIMUM(ODE_eOBJECT)
SETMINIMUM(ODE_eOBJECT)
SHORTTOSTRING(ODE_eOBJECT, ODE_eSTRING, ODE_eOBJECT)
STRINGTOSHORT(ODE_eSTRING, ODE_eSTRING, ODE_eOBJECT)

TabControl

C# support for TabControl
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
TABPAGEDeselcted(ODE_eOBJECT)
TABPAGESelected(ODE_eOBJECT)
Properties
BACKGROUND - has Set method
COUNT
FONT - has Set method
FOREGROUND - has Set method
MULTIPLEROWS - has Set method
ORIENTATION - has Set method
SELECTEDTAB
SELECTEDINDEX
SIZE - has Set method
TABSTOP - has Set method
VISIBLE - has Set method
Methods

ADDPAGE(ODE_eOBJECT)
GETPAGE(ODE_eLONG)
INSERTPAGE(ODE_eLONG, ODE_eOBJECT)
REMOVE(ODE_eOBJECT)
REMOVEAT(ODE_eLONG)
SETENABLEDPAGE(ODE_eLONG, ODE_eBOOL)

Table

C# support for Table
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Events
CELLCLICK (ODE_eOBJECT)
CELDDOUBLECLICK (ODE_eOBJECT)
CELLFOCUSGAINED(ODE_eOBJECT)
CELLFOCUSLOST(ODE_eOBJECT)
CLICK(ODE_eOBJECT)
DATAREQUIRED(ODE_eOBJECT)
ENTERKEYPRESSED(ODE_eOBJECT)
FIELDERROR(ODE_eOBJECT)
FIELDVALIDATION(ODE_eOBJECT)
HEADERCLICK(ODE_eOBJECT)
ROWFOCUSGAINED(ODE_eOBJECT)
ROWFOCUSLOST(ODE_eOBJECT)
Properties
ALTERED - has Set method
AUTOSELECT - has Set method
BACKBUFFER - has Get/Set methods
BACKGRNDCOLOR
BORDER
BORDERSTYLE - has Set method
DATABASESIZE - has Get method
CURRENTCOLUMN
CURRENTROW
EDITABLE - has Set method

ELEVATORPOSITION - has Get method
EMPTY
ERROR
FIRSTVISIBLEROW - has Get/Set methods
FOREGRNDCOLOR
JUSTIFICATION - has Get/Set methods
HEADERBACKGROUND - has Get/Set methods
HEADERFONT - has Get/Set method
HEADERFOREGROUND - has Get/Set methods
HEADERHEIGHT
IMMEDIATEReturn - has Set method
ISAUTOSELECT
ISROWSELECT
LASTVISIBLEROW - has Set method
LINES - has Get/Set methods
MANDATORY - has Set method
NEXTSELECTEDINDEX
NUMBERINGCOLUMN - has Has/Set methods
POPUPMENU - has Set method
ROWHEIGHT
ROWSELECT - has Is/Set methods
SCROLLABLEOCCURS
SCROLLLOCK - has Is/Set method
SCROLLBARS - has Set method
SELECTEDINDEX - has Set method
SELECTEDROWCOUNT - has Get method
SELECTIONMODE - has Get/Set methods
TABSTOP - has Set method
VIEWLINK - has Get/Set methods
VISIBLEOCCURS
Methods
ADDCOLUMN(ODE_eOBJECT)
CLEARSELECTION()
CONVERTTOPHYSICAL(ODE_eLONG)
DISABLETOPANDBOTTOMEVENTS(ODE_eBOOL)
GETCOLUMN(ODE_eLONG)
GETFIRSTVISIBLEOCCURRENCE
GETLASTVISIBLEOCCURRENCE
GETLISTLINK

GETNEXTSELECTEDPHYSICALINDEX
GETNEXTSELECTEDINDEX()
GETNEXTSELECTEDINDEX(ODE_eLONG)
GETOCCURS
GETSCALEDHEADERHEIGHT
GETSCALEDROWHEIGHT
GETSELECTEDPHYSICALINDEX
GETSELECTEDVIRTUALINDEX
GETVISIBLEROWS
NEXTSELECTEDINDEX(ODE_eLONG)
RESETSELECTEDINDEX(ODE_eLONG)
RESETSELECTIONINTERVAL(ODE_eLONG, ODE_eLONG)
SETLISTLINK(ODE_eOBJECT)
SETMOREDATA(ODE_eBOOL)
SETMOREROWS(ODE_eLONG, ODE_eBOOL)
SETNUMBERINGCOLUMN(ODE_eBOOL)
SETSELECTIONINTERVAL(ODE_eLONG, ODE_eLONG)
SETSCALEDHEADERHEIGHT(ODE_eLONG)
SETSCALEDROWHEIGHT(ODE_eLONG)
SETVIRTUALLISTBOXSIZE(ODE_eLONG, ODE_eLONG)

TabPage

C# support for TabPage
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eSTRING)
Properties
DISABLEDIMAGE - has Set method
IMAGE - has Set method
TITLE - has Set method
Methods
ADDCHILD(ODE_eOBJECT)
ADDIMAGE(ODE_eSTRING)

TerminateEvent

C# support for TerminateEvent
Properties
HPSID

Timer

C# support for Timer
Constructors
CONSTRUCTOR(ODE_eOBJECT)
Events
TIMER(ODE_eOBJECT)
Properties
DELAY - has Set method
ENABLED - has Set method
HPSID - has Set method
REPEATS - has Set method
RUNNING
Properties
START()
STOP()

TimerEvent

C# support for TimerEvent
Properties
HPSID
SOURCE

TreeView

C# support for TreeView
Inherits
GuiObject
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eOBJECT)
Events
NODECLICK(ODE_eOBJECT)
NODEDOUBLECLICK(ODE_eOBJECT)

BEFORELAELEDIT(ODE_eOBJECT)
AFTERLAELEDIT(ODE_eOBJECT)
BEFORENODESELECT(ODE_eOBJECT)
AFTERNODESELECT(ODE_eOBJECT)
BEFORENODEEXPAND(ODE_eOBJECT)
AFTERNODEEXPAND(ODE_eOBJECT)
BEFORENODECOLLAPSE(ODE_eOBJECT)
AFTERNODECOLLAPSE(ODE_eOBJECT)
Properties
LAELEDIT - has Get/Set methods
IMAGEINDEX - has Get/Set methods
POPUPMENU - has Set methods
SELECTEDIMAGEINDEX - has Get/Set methods
SELECTEDNODE - has Get/Set methods
TEXT - has Get/Set method
Methods
ADD(ODE_eOBJECT)
ADDBMPTOIMAGELIST(ODE_eSTRING)
CLEAR()
COLLAPSE()
COLLAPSEALL()
COUNT()
EXPAND()
EXPANDALL()
FIND(ODE_eSTRING)
INSERT(ODE_eLONG, ODE_eOBJECT)
REMOVE(ODE_eSTRING)

TreeNode

C# support for TreeNode
Constructors
CONSTRUCTOR()
CONSTRUCTOR(ODE_eOBJECT)
Properties
BACKGROUND - has Set method
HPSID - has Set method
FOREGROUND - has Set method
FONT - has Get/Set method
LAELEDIT - has Get/Set methods

IMAGEINDEX - has Get/Set methods
POPUPMENU - has Set methods
SELECTEDIMAGEINDEX - has Get/Set methods
TEXT - has Get/Set method
Methods
ADD(ODE_eOBJECT)
CLEAR()
COLLAPSE()
COUNT()
EXPAND()
EXPANDALL()
FIND(ODE_eSTRING)
INSERT(ODE_eLONG, ODE_eOBJECT)
REMOVE(ODE_eSTRING)

Window

C# support for Window
Events
CLOSE(ODE_eOBJECT)
CONVERSE(ODE_eOBJECT)
ENTERKEY(ODE_eOBJECT)
INITIALIZE(ODE_eOBJECT)
TERMINATE(ODE_eOBJECT)
WINDOWERROR(ODE_eOBJECT)
WINDOWVALIDATION(ODE_eOBJECT)
Properties
ALTERED - has Is/Set methods
CLIENTSIZE
CLIENTSIZESET - has Is method
DEFAULTBUTTON
FINDFOCUSEDGUIOBJECT
FOCUSEDGUIOBJECT
FOCUSOWNER - has Get method
GETABFMENUBAR
GETDEFAULTPUSHBUTTON
GETSCALEDCLIENTSIZE
HELPTOPIC - has Get method
MAXIMIZABLE - has Is/Set method

MENUBAR
MINIMIZABLE - has Is/Set methods
POPUPMENU - has Get/Set methods
RESIZABLE - has Is/Set methods
TEXT - has Set method
TITLE - has Set method
Methods
ADDCHILD(ODE_eOBJECT)
CLEARALTERED()
CLEARSELECTION()
CLEARWINDOWCHANGES()
PRINTFRAME()
SETABFMENUBAR(ODE_eOBJECT)
SETDEFAULTPUSHBUTTON(ODE_eOBJECT)
SETSCALEDCLIENTSIZE(ODE_eOBJECT)
SHOWMESSAGEBOX(ODE_eSTRING, ODE_eSTRING, ODE_eLONG, ODE_eLONG)
SHOWMESSAGEBOX(ODE_eSTRING, ODE_eLONG)
TERMINATE()
UPDATEDISPLAY()

WindowValidationEvent

C# support for WindowValidationEvent
Properties
ACCEPT
FIELDERROR
HPSID
MANDATORYERROR
SETACCEPT
SETSHOWMESSAGE
SHOWMESSAGE
SOURCE

Sample Code

The sample in this section demonstrates ObjectSpeak for user interface object CheckBox. You can also use the zip file from {Install_Directory}\SAMPLES\Java to import it into the Repository.

Check Box Sample



```

*> -----< *
*> Sample rule to demonstrate ObjectSpeak for GUI control: Check Box < *
*> -----< *
dcl
aBool boolean;
aChar char(5);
booleanChar proc (aBoolean boolean):char(5);
traceGuiObject proc (aGuiObject object type GuiObject);
traceObject proc (anObject object type CheckBox);
clickedObject object type CheckBox;
definePopupMenu proc;
aPopupMenu object type PopupMenu;
aMenuItem object type MenuItem;
MenuClick proc for Click object aMenuItem
(e object type ClickEvent);
aCheckBox object type CheckBox;
checkBoxClick proc for Click type CheckBox
(e object type ClickEvent);
checkBoxFocusGained proc for FocusGained type CheckBox
(e object type FocusGainedEvent);
checkBoxFocusLost proc for FocusLost type CheckBox
(e object type FocusLostEvent);
enddcl

// Initialize the window
proc for Initialize object SAMPLE_CHECKBOX
(e object type InitializeEvent)
// define a popup menu
definePopupMenu()
STATIC_TB.setPopupMenu(aPopupMenu)
// instantiate the object
map new CheckBox
to aCheckBox
// set the properties using the methods provided
aCheckBox.setAltered(false)
aCheckBox.setBackground(new Color(212,208,200))
aCheckBox.setDataLink(SAMPLE_CHECKBOX_FLD2 of SAMPLE_CHECKBOX_W)
aCheckBox.setEnabled(true)
aCheckBox.setFont(Font.Swiss10)
aCheckBox.setForeground(Color.Black)
aCheckBox.setHpsID("DYNAMIC_TB")
aCheckBox.setImmediateReturn(false)
aCheckBox.setLocation(30,50)
aCheckBox.setPopupMenu(aPopupMenu)
aCheckBox.setSelected(false)
aCheckBox.setShortHelp('Dynamic CheckBox linked to a boolean field' )
aCheckBox.setSize(250,16)
aCheckBox.setTabStop(true)
aCheckBox.setText("Dynamic CheckBox")
aCheckBox.setText("abcdefghijklmnopqrstuvwxyz")
// aCheckBox.aCheckBox.setMnemonic('D') // or
aCheckBox.setMnemonicKeyCode(30)
aCheckBox.setVisible(true)
aCheckBox.setFocus()
// define the listener for the object
handler aCheckBox(checkBoxClick)
handler aCheckBox(checkBoxFocusGained)
handler aCheckBox(checkBoxFocusLost)
// add the object to the window
SAMPLE_CHECKBOX.addChild(aCheckBox)
endproc
// process the click event for this type of object
proc checkBoxClick for Click type CheckBox
(e object type ClickEvent)
map thisRule.findGuiObject(e.HpsID)
to clickedObject
trace('-----')
trace('Message:checkBox '+e.HpsID++' has been clicked')
trace('Here are the objects properties:-')
traceGuiObject(e.Source)

```

```

traceObject(clickedObject)
endproc
// process the FocusGained event for this type of object
proc checkBoxFocusGained for FocusGained type CheckBox
(e object type FocusGainedEvent)
map thisRule.findGuiObject(e.HpsID)
to clickedObject
trace('-----')
trace('Message:checkBox '++e.HpsID++' has focus gained')
trace('Here are the objects properties:-')
traceGuiObject(e.Source)
traceObject(clickedObject)
endproc
// process the FocusLost event for this type of object
proc checkBoxFocusLost for FocusLost type CheckBox
(e object type FocusLostEvent)
map thisRule.findGuiObject(e.HpsID)
to clickedObject
trace('-----')
trace('Message:checkBox '++e.HpsID++' has focus lost')
trace('Here are the objects properties:-')
traceGuiObject(e.Source)
traceObject(clickedObject)
endproc
// process the popup menu
proc MenuClick for Click object aMenuItem
(e object type ClickEvent)
endproc
// process the close event
proc for Close object SAMPLE_CHECKBOX
(e object type CloseEvent)
SAMPLE_CHECKBOX.Terminate
endproc
// Terminate the window
proc for Terminate object SAMPLE_CHECKBOX
(e object type TerminateEvent)
endproc

// trace the properties of the GuiObject
proc traceGuiObject(aGuiObject object type GuiObject)
trace(' Altered : '++ booleanChar(aGuiObject.Altered()))
endproc
// trace the properties of the object
proc traceObject(anObject object type CheckBox)
trace(' Background : '
++ char(anObject.Background().getRed()) ++ ', '
++ char(anObject.Background().getGreen()) ++ ', '
++ char(anObject.Background().getBlue()))
trace(' Enabled : '
++ booleanChar(anObject.Enabled()))
trace(' Font : '
++ anObject.Font().displayName())
trace(' Foreground : '
++ char(anObject.Foreground().getRed()) ++ ', '
++ char(anObject.Foreground().getGreen()) ++ ', '
++ char(anObject.Foreground().getBlue()))
trace(' HpsID : '
++ anObject.HpsID())
trace(' ImmediateReturn : '
++ booleanChar(anObject.ImmediateReturn()))
trace(' Location : '
++ char(anObject.Location().X()) ++ ', '
++ char(anObject.Location().Y()))
// trace(' Mnemonic : '++ anObject.Mnemonic())
// trace(' Mnemonic : '++ char(anObject.MnemonicKeycode()))
trace(' Selected : '
++ booleanChar(anObject.Selected()))
trace(' ShortHelp : '
++ anObject.shortHelp())
trace(' Size : '

```

```

++ char(anObject.Size().Width()) ++ ','
++ char(anObject.Size().Height())
trace(' TabStop :')
++ booleanChar(anObject.TabStop())
trace(' Text :')
++ anObject.Text()
trace(' Visible :')
++ booleanChar(anObject.Visible())
trace(' Focus :')
++ booleanChar(anObject.hasFocus())
caseof (anObject.HpsID())
case 'STATIC_TB'
map anObject.dataLink()
to aChar
case 'DYNAMIC_TB'
map anObject.dataLink()
to aBool
map booleanChar(aBool)
to aChar
endcase
trace(' Linked Data :'+aChar)
endproc
// Convert a boolean into character
proc booleanChar(aBoolean boolean):char(5)
if aBoolean
proc return('True')
else
proc return('False')
endif
endproc
// define a popup menu
proc definePopupMenu
map new PopupMenu
to aPopupMenu
map new MenuItem
to aMenuItem
aMenuItem.setHpsID("SAMPLE_MI")
aMenuItem.setText("Sample Popup menu")
aMenuItem.setMnemonic('P')
Handler aMenuItem(MenuClick)
aPopupMenu.add(aMenuItem)
endproc

```