Reference
Version 5.3
September 2008

Construct Spectrum™

Order Number: SPV530-030IBW

This document applies to Construct Spectrum Version 5.3 and to all subsequent releases. Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Readers' comments are welcomed. Comments may be addressed to the Documentation Department at the following e-mail address: Documentation@softwareag.com.

TABLE OF CONTENTS

2. BROWSE CLASSES

7. OBJECT FACTORY

8. PAGE HANDLER CLASSES

11. UTILITY SUBROUTINES ON THE CLIENT

12. SPECTRUM DISPATCH CLIENT INTERNALS

PREFACE

Construct Spectrum Reference is a reference to application programming interfaces (APIs). This preface will help you get the most out of this documentation and find other sources of information about creating and managing Construct Spectrum applications.

The following topics are covered:

Purpose and Structure of this Guide

Construct Spectrum Reference is designed for application developers and administrators who want to quickly find information about Construct Spectrum programming interfaces. It also contains information about utility subroutines on the client and Spectrum Dispatch Client internals.

The following table describes the information contained in each chapter:

Chapter	Title	Topics
1	**Application Preference Classes**, page 23	Describes the classes you can use to implement application preferences in your Construct Spectrum applications.
2	**Browse Classes**, page 49	Describes the Browse object and the browse classes you can use to implement browse functionality in your Construct Spectrum applications.
3	**Business Data Type (BDT) Classes**, page 155	Describes the business data type classes: BDTController and BDTConversion.
4	**Command Handler Classes**, page 183	Describes the command handler classes, as well as their methods and properties.
5	**Maintenance Classes**, page 203	Describes the maintenance classes, which cooperate with a maintenance dialog and other utility subroutines to provide maintenance functionality.
6	**Object Error Classes**, page 247	Describes the object error classes: ObjectError and ObjectError.
7	**Object Factory**, page 263	Describes the object factory module, which identifies and instantiates all objects and methods in an application. This chapter also describes the Open dialog.
8	**Page Handler Classes**, page 275	Describes the page handler classes: ABO and ICSTPageHandler.
9	**Resource Classes**, page 287	Describes the resource classes: Resource and ResourceGroup. These classes provide functions to help internationalize your Construct Spectrum application.

Chapter	Title	Topics (continued)
10	**Spectrum Dispatch Client Classes**, page 301	Describes the Spectrum Dispatch Client (SDC) and its associated classes. It also contains information about each of the classes exposed in the SDC, as well as their methods and properties.
11	**Utility Subroutines on the Client**, page 369	Provides a brief description of the utility routines included in the client framework.
12	**Spectrum Dispatch Client Internals**, page 403	Describes what happens when Natural data is converted to Visual Basic and vice versa for transmission between a server and client.

Document Conventions

This documentation uses the following typographical conventions:

Example	Description
Introduction	Bolded text in cross references indicates chapter and section titles.
"A"	Items within quotation marks indicate values you must enter.
Browse model, GotFocus, Enter	Mixed case text indicates names of: • Natural Construct and Construct Spectrum editors, fields, files, functions, models, panels, parameters, subsystems, variables, and dialogs • Visual Basic classes, constants, controls, dialogs, events, files, menus, methods, properties, and variables • Keys
Alt+F1	A plus sign (+) between two key names indicates that you must press the keys together to invoke a function. For example, Alt+F1 means hold down the Alt key while pressing the F1 key.
CHANGE-HISTORY	Uppercase text indicates the names of Natural command keywords, command operands, data areas, helproutines, libraries, members, parameters, programs, statements, subprograms, subroutines, user exits, and utilities.
Construct Spectrum Reference, *variable name*	Italicized text indicates: • Book titles • Placeholders for information you must supply
[*variable*]	In syntax and code examples, values within square brackets indicate optional items.
{WHILE|UNTIL}	In syntax examples, values within brace brackets indicate a choice between two or more items; each item is separated by a vertical bar (|).

Other Resources

This section provides information about other resources you can use to learn more about Construct Spectrum and Natural Construct. For more information about these documents and courses, contact the nearest Software AG office or visit the website at www.softwareag.com to order documents or view course schedules and locations. You can also use the website to email questions to Customer Support.

Related Documentation

This section lists other documentation in the Construct Spectrum and Natural Construct documentation set.

Construct Spectrum

- *Construct Spectrum SDK Reference*
  This documentation is for developers creating Natural modules and ActiveX Business Objects to support applications that will run in the Natural mainframe environment and a Windows environment and/or an internet server.
- *Construct Spectrum SDK for Web Applications*
  This documentation is for developers creating the web components of applications. It describes how to use the Construct Spectrum wizards in Visual Basic to generate HTML templates, page handlers, and object factory entries. It also contains detailed information about customizing, debugging, deploying, and securing web applications.
- *Construct Spectrum SDK for Client/Server Applications*
  This documentation is for developers creating client components for applications that will run in a Natural mainframe (server) and Windows (client) environment.
- *Construct Spectrum Messages*
  This documentation is for application developers, application administrators, and system administrators who want to investigate messages returned by Construct Spectrum runtime and SDK components.

Natural Construct

- *Natural Construct Installation Guide for Mainframes*
  This documentation provides essential information for setting up the latest version of Natural Construct, which is needed to operate the Construct Spectrum programming environment.
- *Natural Construct Generation*
  This documentation describes how to use the Natural Construct models to generate applications that will run in a mainframe environment.
- *Natural Construct Administration and Modeling*
  This documentation describes how to use the Administration subsystem of Natural Construct and how to create new models.
- *Natural Construct Help Text*
  This documentation describes how to create online help for applications that run on server platforms.
- *Natural Construct Getting Started Guide*
  This guide introduces new users to Natural Construct and provides step-by-step instructions to create several common processes.

Other Documentation

This section lists documents published by WH&O International:

- *Natural Construct Tips & Techniques*
  This book provides a reference of tips and techniques for developing and supporting Natural Construct applications.
- *Natural Construct Application Development User's Guide*
  This guide describes the basics of generating Natural Construct modules using the supplied models.
- *Natural Construct Study Guide*
  This guide is intended for programmers who have never used Natural Construct.

Related Courses

In addition to the documentation, the following courses are available from Software AG:

- A self-study course on Natural Construct fundamentals
- An instructor-led course on building applications with Natural Construct
- An instructor-led course on modifying the existing Natural Construct models or creating your own models

APPLICATION PREFERENCE CLASSES

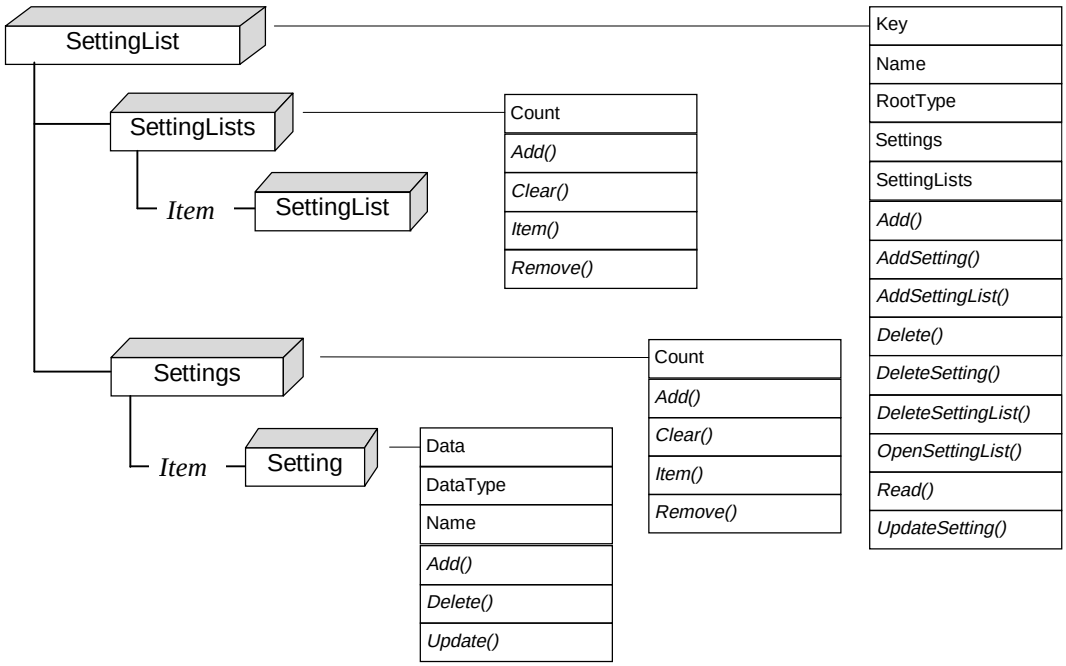
This chapter describes the classes you can use to implement application preferences in your Construct Spectrum applications: Setting, Settings, SettingList, and SettingLists (supplied in the CSTVBFW.dll client framework component). GUI applications composed of widely distributed application components frequently require the ability to write and read settings that must remain constant during multiple executions of the application (for example, presentation preferences or communication settings).

Use the application preference classes to add, read, and update:

- User preferences (settings that tailor an application to an individual user, such as an option to remember a password or a list of recently opened documents)
- Application preferences (settings that apply to the application itself, irrespective of the users, such as the file and path for the help file used by the application)

The following topics are covered:

The following diagram shows the structure of the application preference classes:



Structure of the Application Preference Classes

These methods and properties are described in the following sections.

Understanding Application Preferences

Before defining application preferences, you should understand the architecture of the settings. These settings are hierarchical in nature; each Setting object belongs to a SettingList object (similar to a file in a directory). And each SettingList object contains a collection of Setting objects and SettingLists objects.

Each Setting object is identified by a name and a value type. You can assign the following value types to a setting:

Setting Value Type	Description
String	Visual Basic string.
Binary	Binary data in any form; the default is 0.
DoubleWord	32-bit number; the default is 0.

Using Constants as Setting Value Types

This section describes the constants you can use as setting values. These constants are public and supplied in the CSTObjectConstants.bas client framework component.

You can use the following constants as value types for the Setting class:

Constant	Value	Description
SETTING_SZ	1	Null terminated string.
SETTING_BINARY	3	Free form binary value.
SETTING_DWORD	4	32-bit number.

You can use the following constants as root value types for the SettingList class:

Constant	Value	Description
SETTING_APPLICATION	&H80000002	Application setting list.
SETTING_USER	&H80000001	User setting list.

Use the SETTING_ROOTKEY constant to prefix all values for the Setting and SettingList classes. This enforces a standard starting location within the settings storage media for all the settings used by an application. For example:

```
SETTING_ROOTKEY = "Software\Spectrum Frameworks"
```

Setting Class

The Setting class creates and manipulates an individual application Setting object. This class has the following methods and properties:

Add Method, page 27	**Delete Method**, page 29
Data Property, page 28	**Name Property**, page 30
DataType Property, page 28	**Update Method**, page 30

Add Method

This method adds a new Setting object to the settings storage media. If a Setting object currently exists with the specified Name, it is updated with lDataType and szData (if successful, this method returns True).

Syntax

object.Add(*szName*, *szKey*, *lRootType*, *lDataType*, *szData*)

Part	Required	Data Type	Description
szName	yes	string	Name of the Setting object.
szKey	yes	string	Hierarchical key used to access the Setting object. This key consists of a series of setting list names concatenated using a backslash (\) character (similar to a path name in a directory). For more information, see **Key Property**, page 41.
lRootType	yes	long	Prefix for the Setting object key (either SETTING_APPLICATION or SETTING_USER).
lDataType	yes	long	Data type for the Setting object (can be SETTING_STRING, SETTING_BINARY, or SETTING_DWORD).
szData	yes	string	Data value for the Setting object.

Remarks

A Setting object is a member of a Settings class, and a Settings class object is a member of a SettingList class. To manipulate a SettingList object, use the AddSetting method for the SettingList class instead of this method. In addition to adding the setting, AddSetting updates the Settings class for the SettingList class.

Example

```
Dim Setting As New Setting
If Setting.Add(settingName, settingKey, privRootType, lDataType, _
    szData) Then
    ' Add this Setting to the collection.
    privSettings.Add Setting
End If
```

See Also

AddSetting Method, page 36.

Data Property

This property identifies the data value assigned to an application Setting class object; the data type can be SETTING_STRING, SETTING_BINARY, or SETTING_DWORD.

Syntax

object.data

DataType Property

This property identifies the type of data assigned to an application Setting class object; the data type must be one of the following:

- SETTING_STRING
- SETTING_BINARY
- SETTING_DWORD

Delete Method

This method deletes a Setting object from the settings storage media (if successful, this method returns True).

Syntax

object.Delete(*szName*, *szKey*, *lRootType*)

Part	Required	Data Type	Description
szName	yes	string	Name of the Setting object.
szKey	yes	string	Hierarchical key used to access the Setting object. This key consists of a series of setting list names concatenated using a backslash (\) character (similar to a path name in a directory). For information, see **Key Property**, page 41.
lRootType	yes	long	Prefix for the Setting object key (either SETTING_APPLICATION or SETTING_USER).

Remarks

If you are deleting a Setting object from the SettingList class, use the DeleteSetting() method for the SettingList class instead of this method.

Example

```
Setting.Delete settingName, settingKey, privRootType
```

See Also

DeleteSetting Method, page 39.

Name Property

This property identifies the setting name. The setting name can contain any character except a backslash (\).

Syntax

object.name

Update Method

This method updates the value and data type for a Setting object in the settings storage media (if successful, this method returns True).

Syntax

object.Update(*lDataType*, *szData*)

Part	Required	Data Type	Description
lDataType	no	long	Data type for the Setting object (can be SETTING_STRING, SETTING_BINARY, or SETTING_DWORD).
szData	no	string	Data value for the Setting object.

Remarks

Before using this method, the Setting object must have been properly initialized (for example, a Setting object returned by the Item method for the Settings class).

If you are updating a Setting object with a reference to a SettingList class, use the UpdateSetting method for the SettingList class instead of this method.

Example

```
Setting.Update szDataType, szData
```

See Also

UpdateSetting Method, page 45.

Settings Class

The Settings class creates and manipulates a collection of Setting objects. This class has the following methods and properties:

Add Method, page 31	**Item Method**, page 33
Clear Method, page 32	**Remove Method**, page 34
Count Property, page 32	

Add Method

This method adds a new Setting object to the internal collection for the Settings class (if successful, this method returns True).

Syntax

object.Add(*setting*)

Part	Required	Data Type	Description
object	yes	Setting	Setting object.
setting	yes	Setting	Reference to a fully initialized Setting object.

Remarks

This method only adds a Setting object to the internal collection for the Settings class — it does not affect the settings storage media.

Example

```
Settings.Add (setting)
```

Clear Method

This method removes all Setting objects from the internal collection for the Settings class.

Syntax

object.Clear

Remarks

This method only removes the Setting objects from the Settings class — it does not affect the settings storage media.

Example

```
Settings.Clear
```

Count Property

This property returns the number of Setting objects in the internal collection for the Settings class.

Syntax

object.Count

Example

```
Settings.Count
```

Item Method

This method returns a specific Setting object from the internal collection for the Settings class, either by position or by setting name.

Syntax

object.Item(*index*)

Part	Required	Data Type	Description
index	yes	variant	If a numeric expression, the index must be a number from 1 to the value of the Count property for the Setting object. If a string expression, the index must correspond to a Name property for a Setting object that is a member of the Settings class.

Example

```
Set Setting = Settings.Item index
```

Remove Method

This method removes a Setting object from the internal collection for the Settings class (if successful, this method returns True).

Syntax

object.Remove(*index*)

Part	Required	Data Type	Description
index	yes	variant	If a numeric expression, the index must be a number from 1 to the value of the Count property for the Setting object. If a string expression, the index must correspond to a Name property for a Setting object that is a member of the Settings class.

Remarks

This method only removes a Setting object from the Settings class — it does not affect the settings storage media.

Example

```
Settings.Remove (index)
```

SettingList Class

The SettingList class creates and manipulates SettingList objects. A SettingList object is an aggregate consisting of a Settings object and a SettingLists object.

The SettingList class has the following methods and properties:

Add Method, page 35	**Name Property**, page 41
AddSetting Method, page 36	**OpenSettingList Method**, page 42
AddSettingList Method, page 37	**Read Method**, page 43
Delete Method, page 38	**RootType Property**, page 44
DeleteSetting Method, page 39	**SettingLists Property**, page 44
DeleteSettingList Method, page 40	**Settings Property**, page 44
Key Property, page 41	**UpdateSetting Method**, page 45

Add Method

This method adds a SettingList object to the settings storage media (if successful, this method returns True).

Syntax

object.Add(*szName*, *szKey*, *lRootType*)

Part	Required	Data Type	Description
szName	yes	string	Name of the SettingList object. For more information, see **Name Property**, page 30.
szKey	yes	string	Hierarchical key used to access the SettingList object. This key consists of a series of setting list names concatenated using a backslash (\) character (similar to a path name in a directory).
lRootType	yes	long	Prefix for the setting list key (either SETTING_APPLICATION or SETTING_USER).

Remarks

If the specified SettingList object currently exists, this method opens that object.

To initialize the new SettingList object, this method populates the Settings and SettingLists classes for the SettingList object.

Example

```
Dim Prefs As New SettingList

' Open or Create the Object Preferences List.
Prefs.Add DataObject.Name, BROWSE_OBJECT_ROOTKEY, SETTING_USER
```

See Also

AddSettingList Method, page 37.

AddSetting Method

This method adds a Setting object to the SettingLists class and to the settings storage media (if successful, this method returns True).

Syntax

object.AddSetting(*szName*, *lDataType*, *szData*)

Part	Required	Data Type	Description
szName	yes	string	Name of the SettingList object. For more information, see **Name Property**, page 41.
lDataType	yes	long	Data type for the Setting object (can be SETTING_STRING, SETTING_BINARY, or SETTING_DWORD).
szData	yes	string	Data value for the Setting object.

Remarks

Before you can use this method, the SettingList object must have been successfully initialized with either an Add() or Read() method (if the SettingList object has not been initialized, this method returns False).

If the Setting object currently exists in the settings storage media, it is updated with the specified data and data type.

Example

```
Dim Prefs As New SettingList

' Open or Create the Browse Object's Preferences List.
If Prefs.Add(DataObject.Name, BR_OBJ_ROOTKEY, SETTING_USER) Then
  With Prefs
    ' Add the browse object's Logical Key setting.
    If .AddSetting("Logical Key", SETTING_SZ, DataObject.SearchKey) _
Then
      PreferencesAdded = PreferencesAdded + 1
    End If
  End With
End If
```

See Also

Add Method, page 27.

AddSettingList Method

This method adds a SettingList object to the SettingLists class and to the settings storage media (if successful, this method returns True).

Syntax

object.AddSettingList(*szName*)

Part	Required	Data Type	Description
szName	yes	string	Name of the SettingList object.

Remarks

Before you can use this method, the SettingList object must have been successfully initialized with either an Add() or Read() method. If the SettingList object has not been initialized, this method returns False.

If the SettingList object currently exists in the setting storage, the method still returns True and no changes are made to the settings storage media.

Example

```
Dim Prefs As New SettingList

' Open or Create the Browse Object's Preferences List.
If Prefs.Add(DataObject.Name, BR_OBJ_ROOTKEY, SETTING_USER) Then
  With Prefs
    ' Add the browse object's Columns Preferences.
    .AddSettingList "Column Preferences"
  End With
End If
```

See Also

Add Method, page 27.

Delete Method

This method deletes a SettingList object from the settings storage media (if successful, this method returns True).

Syntax

object.Delete(*szName*, *szKey*, *lRootType*)

Part	Required	Data Type	Description
szName	yes	string	Name of the SettingList object. For more information, see **Name Property**, page 30.
szKey	yes	string	Hierarchical key used to access the SettingList object. This key consists of a series of setting list names concatenated using a backslash (\) character (similar to a path name in a directory).
lRootType	yes	long	Prefix for the setting list key (either SETTING_APPLICATION or SETTING_USER).

Remarks

This method also deletes all SettingLists objects and Settings objects below this SettingList object in the setting storage hierarchy.

Example

```
Dim Prefs As New SettingList

' Delete the Object Preferences List.
Prefs.Delete DataObject.Name, BROWSE_OBJECT_ROOTKEY, SETTING_USER
```

See Also

DeleteSettingList Method, page 40, and **Remove Method**, page 34.

DeleteSetting Method

This method deletes a Setting object from the SettingLists class and from the settings storage media (if successful, this method returns True).

Syntax

object.DeleteSetting(*szName*)

Part	Required	Data Type	Description
szName	yes	string	Name of the Setting object.

Remarks

Before you can use this method, the SettingList object must have been successfully initialized with either an Add() or Read() method. If the SettingList object has not been initialized, this method returns False.

If the Setting object does not currently exist in the setting storage, this method returns True.

Example

```
Dim Prefs As New SettingList

' Open or Create the Browse Object's Preferences List.
If Prefs.Read(DataObject.Name, BR_OBJ_ROOTKEY, SETTING_USER) Then
  With Prefs
    ' Delete the browse object's Logical Key setting.
    .DeleteSetting("Logical Key") Then
  End With
End If
```

See Also

Delete Method, page 38, and **Remove Method**, page 48.

DeleteSettingList Method

This method deletes a SettingList object from the SettingLists class and from the settings storage media (if successful, this method returns True).

Syntax

object.DeleteSettingList(*szName*)

Part	Required	Data Type	Description
szName	yes	string	Name of the SettingList object.

Remarks

Before you can use this method, the SettingList object must have been successfully initialized with either an Add() or Read() method. If the SettingList object has not been initialized, this method returns False.

If the SettingList object does not currently exist in the setting storage, this method returns True.

Example

```
Dim Prefs As New SettingList

' Open or Create the Browse Object's Preferences List.
If Prefs.Read(DataObject.Name, BR_OBJ_ROOTKEY, SETTING_USER) Then
  With Prefs
    ' Delete the browse object's Columns settings.
    .DeleteSettingList("Columns") Then
  End With
End If
```

See Also

Remove Method, page 48.

Key Property

This property identifies the hierarchical key used to access the SettingList object. The key consists of a series of setting list names concatenated using the backslash (\) character, similar to a path name in a directory structure. Each SettingList object is considered a “child” of the SettingList object that preceded it.

For example, “ACME Order Entry\Browse Preferences\Order Browse\” indicates that this SettingList object is at least three setting lists deep in the settings hierarchy. The value of the RootType property determines how deep in the hierarchy it is, because the root type may add several more layers of setting lists above the first setting list in the key.

If the key is null, the SettingList object is located directly under the SettingList object identified by the RootType property.

When a Setting or SettingList object is added to the settings storage and the Key property specifies SettingList objects that have not yet been created, the specified objects are created.

Syntax

object.Key

Example

```
SettingList.Key = “Acme ...”
```

Name Property

This property identifies the name of the SettingList object. The name can contain any character except a backslash (\).

Syntax

object.Name

Example

```
SettingList.Name = “Column”
```

OpenSettingList Method

This method opens a SettingList object from the SettingLists class (if successful, this method returns an initialized reference to a SettingList object).

Syntax

object.OpenSettingList(*szName*)

Part	Required	Data Type	Description
szName	yes	string	Name of the SettingList object.

Remarks

Before you can use this method, the SettingList object must have been successfully initialized with either an Add() or Read() method. If the SettingList object has not been initialized, this method returns False.

If the SettingList object does not exist in the setting storage, nothing is returned.

Example

```
Dim Prefs As New SettingList
Dim ColumnPrefs As SettingList
' Open or Create the Browse Object's Preferences List.
If Prefs.Read(DataObject.Name, BR_OBJ_ROOTKEY, SETTING_USER) Then
  With Prefs
    ' Open the browse object's Columns settings.
    ColumnPrefs = .OpenSettingList "Columns"
  End With
End If
```

See Also

Add Method, page 27, **Read Method**, page 43, and **Item Method**, page 47.

Read Method

This method reads an existing SettingList object from the settings storage media (if successful, this method returns True; if the SettingList object does not exist, this method returns False).

Syntax

object.Read(*szName*, *szKey*, *lRootType*)

Part	Required	Data Type	Description
szName	yes	string	Name of SettingList object. For more information, see **Name Property**, page 30.
szKey	yes	string	Hierarchical key used to access the SettingList object. This key consists of a series of setting list names concatenated using a backslash (\) character (similar to a path name in a directory).
lRootType	yes	long	Prefix for the setting list key (either SETTING_APPLICATION or SETTING_USER).

Remarks

To initialize a new SettingList object, this method populates the Settings and SettingLists classes for the SettingList object.

Example

```
Dim Prefs As New SettingList

' Read the Object Preferences List.
Prefs.Read DataObject.Name, BROWSE_OBJECT_ROOTKEY, SETTING_USER
```

See Also

Add Method, page 27.

RootType Property

This property defines the Key property prefix for the SettingList object; valid values are SETTING_APPLICATION or SETTING_USER.

The RootType property determines how to access the setting list in the settings storage media. It also indicates whether the SettingList object contains settings specific to an application or specific to individual users.

Syntax

object.RootType

SettingLists Property

This property references a SettingLists class object (an object containing a collection of all the SettingLists objects that are linked directly to the SettingList object).

Syntax

object.SettingLists

Example

```
Set SettingLists = SettingList.SettingLists
```

Settings Property

This property references a Settings class object (an object containing a collection of all the Settings objects that are linked directly to the SettingList object).

Syntax

object.Settings

Example

```
Set Settings = SettingList.Settings
```

UpdateSetting Method

This method updates a Setting object in the SettingList class and in the settings storage media (if successful, this method returns True).

Syntax

object.UpdateSetting(*szName*, *lDataType*, *szData*)

Part	Required	Data Type	Description
szName	yes	string	Name of a Setting object.
lDataType	no	long	Data type for the Setting object (can be SETTING_STRING, SETTING_BINARY, or SETTING_DWORD).
szData	no	string	Data value for the Setting object.

Remarks

Before you can use this method, the SettingList object must have been successfully initialized with either an Add() or Read() method. If the SettingList object has not been initialized, this method returns False.

If the Setting object does not exist in the setting storage, this method returns False.

Example

```
Dim Prefs As New SettingList

' Open or Create the Browse Object's Preferences List.
If Prefs.Add(DataObject.Name, BR_OBJ_ROOTKEY, SETTING_USER) Then
  With Prefs
    ' Update the browse object's Logical Key setting.
    .UpdateSetting "Logical Key", SETTING_SZ, DataObject.SearchKey
  End With
End If
```

See Also

AddSetting Method, page 36.

SettingLists Class

The SettingLists class creates and manipulates a collection of SettingList objects. A SettingList object is a collection of Setting objects.

The SettingList class has the following methods and properties:

Add Method, page 27	**Item Method**, page 33
Clear Method, page 32	**Remove Method**, page 48
Count Property, page 32	

Add Method

This method adds a new SettingList object to the internal collection for the SettingLists class (if successful, this method returns True).

Syntax

object.Add(*settinglist*)

Part	Required	Data Type	Description
settinglist	yes	Setting	Reference to a fully initialized SettingList object.

Remarks

This method only adds a SettingList object to the internal collection for the SettingLists class — it does not affect the settings storage media.

Example

```
SettingLists.Add (SettingList)
```

Clear Method

This method removes all SettingList objects from the internal collection for the SettingLists class.

Syntax

object.Clear

Remarks

This method only clears the SettingLists class — it does not affect the settings storage media.

Count Property

This property returns the number of SettingLists objects in the internal collection for the SettingLists class.

Syntax

object.Count

Item Method

This method returns a specified SettingList object from the internal collection for the SettingLists class, either by position or by setting list name.

Syntax

object.Item(*index*)

Part	Required	Data Type	Description
index	yes	variant	If a numeric expression, the index must be a number from 1 to the value of the Count property for the SettingLists object. If a string expression, the index must correspond to a Name property for a SettingList object that is a member of the SettingLists class.

Remove Method

This method removes a SettingList object from the internal collection for the SettingLists class (if successful, this method returns True).

Syntax

object.Remove(*index*)

Part	Required	Data Type	Description
index	yes	variant	If a numeric expression, the index must be a number from 1 to the value of the Count property for the SettingLists object. If a string expression, the index must correspond to a Name property for a SettingList object that is a member of the SettingLists class.

Remarks

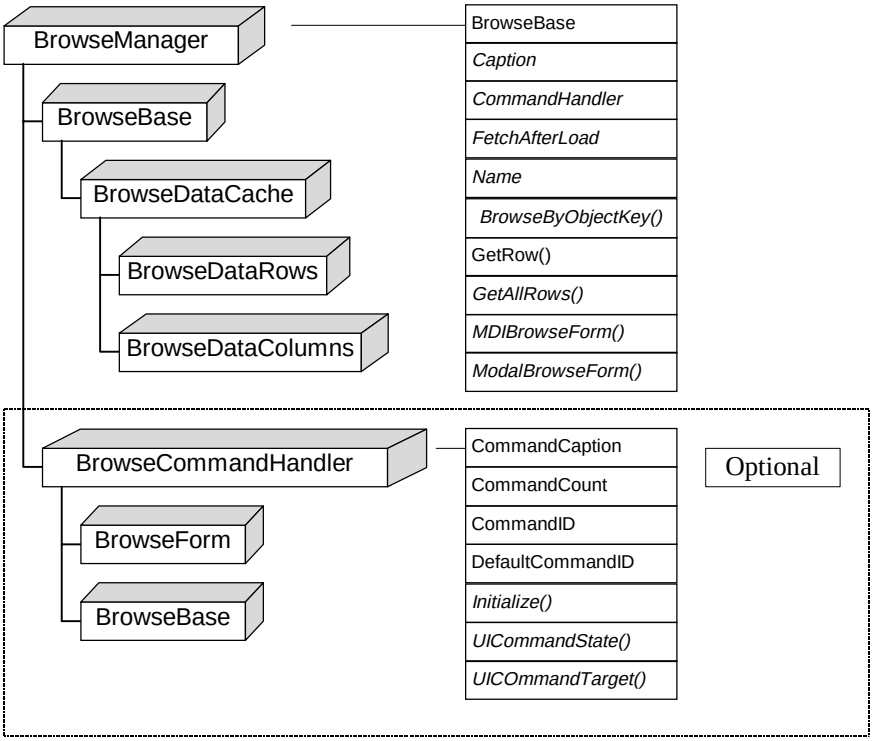
This method only removes a SettingList object from the internal collection for the SettingLists class — it does not affect the settings storage media.

BROWSE CLASSES

This chapter describes the Browse object and the browse classes you can use to implement browse functionality in your Construct Spectrum applications.

The following topics are covered:

The following diagram shows the structure of the browse classes:



Structure of the Browse Classes

These methods and properties are described in the following sections.

Browse Object

This section introduces the Browse object and includes a description of each of the public methods and properties used by a Browse object.

The Browse object contains only enough code to define data columns, logical search keys, and a few other parameters necessary to allow communication with the server. The bulk of the Browse object's work is done by one of its components: the BrowseBase object. After an instance of the Browse object is created, a reference to the BrowseBase class is passed back to the application and is used in place of the Browse object thereafter.

The Browse object has the following methods and properties:

BrowseBase Property, page 51	**InitKeys Method**, page 52
InitColumns Method, page 52	

BrowseBase Property

This property gets the BrowseBase property value for the specified Browse object.

Syntax

```
[browsebase_object] = [business_object].BrowseBase
```

Remarks

The Browse object has only one public property, an instance of the BrowseBase class, which it creates and initializes with its defining characteristics. These characteristics are defined by and derived from Construct Spectrum and consist of data areas, logical search key values, and some display parameter information.

Example

```
Dim DataObject as BrowseBase
Set DataObject = Customers.BrowseBase
```

Data Type

BrowseBase object

See Also

BrowseBase Class, page 53.

InitColumns Method

This method initializes column definitions for the Browse object.

Syntax

```
[business_object].InitColumns
```

Remarks

To change the order in which columns are displayed on a browse dialog, modify the code Construct Spectrum generates for this method.

Example

```
Customers.InitColumns
```

See Also

InitKeys Method, page 52.

InitKeys Method

This method initializes the logical search key definitions for the Browse object.

Syntax

```
[business_object].InitKeys
```

Remarks

This method initializes an internal collection of logical names for keys that are defined prior to generation and are used to make browse requests to the server. Each key contains a collection of key fields that are used to build a browse request; only compound keys have more than one key field in any collection. Each logical key has one of these collections. This code is generated by Construct Spectrum and should not be changed.

Example

```
Customers.InitKeys
```

See Also

InitColumns Method, page 52.

BrowseBase Class

This class instantiates BrowseBase objects, which you can use to manage interactions between the remote Browse object subprogram and the object's clients (which include generated applications and, indirectly, users via a browse dialog). This class has methods for storing and retrieving logical search key definitions, database formatting information, presentation-level settings, and the results of server data requests.

The BrowseBase class has the following methods and properties:

Cache Property

This property accesses the data cache for the Browse object.

Syntax

```
[baseobject].Cache.Method_orProperty
```

Remarks

Use this property to cache data retrieved from the data source or pre-loaded during initialization.

Example

```
With DataObject.Cache
     If .Rows.SelectedCount > 0 Then
          BrowseByObjectKey = True
     End If
End With
```

Data Type

BrowseDataCache object

CacheInsertData Method

This method inserts an array of data from an outside data source into the data cache for the Browse object.

Syntax

```
[baseobject].CacheInsertData(unique_row_id, row_fields)
```

Part	Required	Data Type	Description
unique_row_id	yes	string	Unique ID for the data row.
row_fields	yes	variant ParamArray	Array containing the column data.

Remarks

This method is used internally by business objects containing static data to preload the BrowseBase class data cache during initialization. The data row must have all the attributes of a normal Adabas row, including the UNIQUE-ID field.

Example

```
With BaseObj
     If .StaticData Then
       ' Load object data from the file input array.
       .CacheInsertData(unique_row_id, row_fields)
     End If
End With
```

CacheRefresh Method

This method refreshes the cached data with the equivalent current data (using the current search key value) by making a fetch request to the Browse object on the server.

Syntax

[*baseobject*].CacheRefresh

Remarks

Use this method when you believe the currently cached data is stale or no longer valid.

Example

```
With BaseObj
     If .StaticData Then
       ' Load object data from the file input array.
       .CacheRefresh
     End If
End With
```

Compress Property

This property gets or sets the Compress flag for the BrowseBase object.

Syntax

[*baseobject*].Compress = [{True | False}]

Remarks

The Compress flag indicates whether data compression should be performed when sending data to the server. In most cases, this is negligible because the only data sent is the browse key value and a private data PDA containing only a small amount of data.

Example

```
With BaseObj
     .Compress = True
End With
```

Data Type

Integer (Boolean)

See Also

Encrypt Property, page 56.

Encrypt Property

This property gets or sets the Encrypt flag for the BrowseBase object.

Syntax

[*baseobject*].Encrypt = [{True | False}]

Remarks

The Encrypt flag indicates whether data encryption should be performed when sending data to the server.

Example

```
With BaseObj
     .Encrypt = True
End With
```

Data Type

Integer (Boolean)

See Also

Compress Property, page 55.

EndOfData Property

This property gets or sets the End of Data flag for the BrowseBase object.

Syntax

```
[baseobject].EndOfData = [{True | False}]
```

Remarks

Typically, the End of Data flag is set as a result of the server's action. You can test this flag to determine when all data matching the search parameters has been returned.

Example

```
With BaseObj
     ' Restart query after end of data
     If .EndOfData Then
          .Restart = True
     End If
End With
```

Data Type

Integer (Boolean)

See Also

Restart Property, page 61, and **RowsFetched Property**, page 62.

Fetch Method

This method sends a data request to the server.

Syntax

```
[baseobject].Fetch [fetch_count]
```

Part	Required	Data Type	Description
fetch_count	no	Integer	Number of data records to fetch.

Remarks

Calls the Spectrum Dispatch Client to reach the server and return fetch count elements.

Example

```
With BaseObj
     While Not .EndOfData
          ' Load 10 more data records from server.
          .Fetch 10
     Wend
End With
```

FetchAll Method

This method fetches all matching data from the server.

Syntax

[*baseobject*].FetchAll

Remarks

Calls the Spectrum Dispatch Client to reach the server until all data matching the search key is returned.

Example

```
With BaseObj
     ' Load all matching data from server.
     .FetchAll
End With
```

LoadObjectPreferences Method

This method reads preferences stored in the Windows registry; it uses these preferences to initialize some settings in the BrowseBase object. If all settings are read successfully, this method returns True.

Syntax

[*baseobject*].LoadObjectPreferrences

Remarks

For a specified Browse object (identified by the Name property), this method reads initial settings from the Windows registry, such as:

- logical search key name
- fixed search key components
- range option
- use distinct key setting

Example

```
BrowseBase.LoadObjectPreferrences
```

Data Type

Boolean

See Also

SaveObjectPreferences Method, page 62.

Name Property

This property gets or sets the name of the associated generated Browse object, which is used to initialize and configure a BrowseBase object.

Syntax

```
[baseobject].Name
```

Remarks

This name is used to check security privileges in the Spectrum security service; the text is normally generated by Construct Spectrum.

Example

```
With BaseObj
     ' Use the name of the object as a caption
     caption = .Name
End With
```

Data Type

String

RangeOption Property

This property gets or sets the range option for the logical selection key.

Syntax

`[baseobject].RangeOption`

Remarks

Displayed Text	Description	Constant
*	Embedded wildcard	BR_EMBEDDED_WILDCARD_RO
<	Less than	BR_LESS_THAN_RO
<=	Less than or equal	BR_LESS_OR_EQUAL_RO
=	Equal	BR_EQUAL_RO
>=	Greater than or equal	BR_GREATER_OR_EQUAL_RO
>	Greater than	BR_GREATER_RO
ABC	Begins with	BR_BEGINS_WITH_RO
blank	No wildcard	BR_NO_WILDCARD_RO

Example

```
' Get the current range option.
With BaseObj
     CurrentRange = .RangeOption
End With
```

Data Type

Integer

See Also

SearchKey Property, page 63, **SearchKeyFixedComponents Property**, page 64, and **UseUniqueID Property**, page 67.

Reset Method

This method resets the Browse object.

Syntax

```
[baseobject].Reset
```

Remarks

This method initializes the cache areas to their original states.

Example

```
With BaseObj
     .Reset
End With
```

Restart Property

This property gets or sets the Restart Search flag for the Browse object.

Syntax

```
[baseobject].Restart = [{True | False}]
```

Remarks

To indicate the start of a new search, set the Restart Search flag.

Example

```
With BaseObj
     ' Restart query after end of data
     If .EndOfData Then
          .Restart = True
     End If
End With
```

Data Type

Integer (Boolean)

See Also

EndOfData Property, page 57, and **RowsFetched Property**, page 62.

RowsFetched Property

This property gets the total number of rows returned from the server for the current search key request.

Syntax

```
count = [baseobject].RowsFetched
```

Remarks

This number is limited to a range between 1 and the maximum number of rows defined for the object.

Example

```
Cache.RowsFetched
```

Data Type

Integer

See Also

EndOfData Property, page 57, and **Restart Property**, page 61.

SaveObjectPreferences Method

This method saves property settings used by an instance of a BrowseBase class in the Windows registry. If all settings are saved successfully, this method is True.

Syntax

```
[baseobject].SaveObjectPreferrences
```

Remarks

For a specified Browse object (identified by the Name property), this method saves the following settings in the Windows registry:

- search key name
- range option
- search key fixed components
- use distinct key

Example

```
BrowseBase.SaveObjectPreferrences
```

Data Type

Boolean

See Also

LoadObjectPreferences Method, page 58.

SearchKey Property

This property gets or sets the name of the current logical search key.

Syntax

```
key_name = [baseobject].SearchKey
```

Remarks

The key must be a member of the search keys collection. This collection is created and initialized when BrowseBase is instantiated by a generated Browse object.

Example

```
With BaseObj
     ' Set a new search key
     .SearchKey = "CUSTOMER-WAREHOUSE-ID"
End With
```

Data Type

String

See Also

- **RangeOption Property**, page 60
- **SearchKeyFixedComponents Property**, page 64
- **UseDistinctKey Property**, page 66
- **UseUniqueID Property**, page 67

SearchKeyFixedComponents Property

This property gets or sets the current number of leading fixed-key fields used by the current logical search key.

Syntax

```
count = [baseobject].SearchKeyFixedComponents
```

Remarks

Use this property to “fix” or “lock” certain field components of a search key. Browse dialogs use this property to enable or disable the editing of key fields.

Key fields are locked from left to right, where the leftmost key field is the most significant component of a compound key. This property specifies the number of key fields to lock, starting with the leftmost key field.

Example

```
With BaseObj
     ' Set a new fixed field count
     .SearchKeyFixedComponents = 1
End With
```

Data Type

Integer

See Also

SearchKey Property, page 63, **UseDistinctKey Property**, page 66, and **UseUniqueID Property**, page 67.

SearchKeyValue Property

This property gets or sets the value of a key field used by the current search key.

Syntax

```
SearchKeyValue(keyfield) = value
```

Part	Required	Data Type	Description
keyfield	yes	string	Name of the field.
value	yes	variant	Data match requirement.

Remarks

A search key may use several key fields — which may themselves be used by several keys.

Example

```
With BaseObj
     ' Set a new search key (blank the old Customer value)
     If .RowsFetched = 0 Then
          .SearchKeyValue("Customers") = ""
     End If
End With
```

Data Type

Variant

StaticData Property

This property gets or sets a flag indicating that the browse cache contains static data.

Syntax

```
[baseobject].StaticData = {True | False}
```

Remarks

Use this property in conjunction with objects containing fixed or static lists. If this property is True, all access to the remote database is disabled and all search key-related functionality is disabled.

Example

```
With BaseObj
     If Not .StaticData Then
          ' Set a search key.
          .SearchKey = KeyName
     End If
End With
```

Data Type

Integer (Boolean)

UseDistinctKey Property

This property gets or sets the Search Keys Only property.

Syntax

```
[baseobject].UseDistinctKey = {True | False}
```

Remarks

This property corresponds to histogram mode in Natural. If this property is True, the search request made to the server only returns the search key column data.

Example

```
With BaseObj
     If Not .StaticData Then
          ' Set the search to HISTOGRAM mode.
          .UseDistinctKey = True
     End If
End With
```

Data Type

Integer (Boolean)

See Also

RangeOption Property, page 60, **SearchKey Property**, page 63, and **UseUniqueID Property**, page 67.

UseUniqueID Property

This property gets or sets the UseUniqueID flag.

Syntax

```
[baseobject].UseUniqueID = {True | False}
```

Remarks

If this property is True, a search request to the server can be positioned to start at a unique key ID. For example, if the search key is “LastName”, setting this property to True ensures that browsing through a range of “Smiths” repositions the browse correctly on successive Fetch requests.

Example

```
With BaseObj
     If Not .StaticData Then
          ' Set the search to use the UNIQUE-ID field.
          .UseUniqueID = True
     End If
End With
```

Data Type

Integer (Boolean)

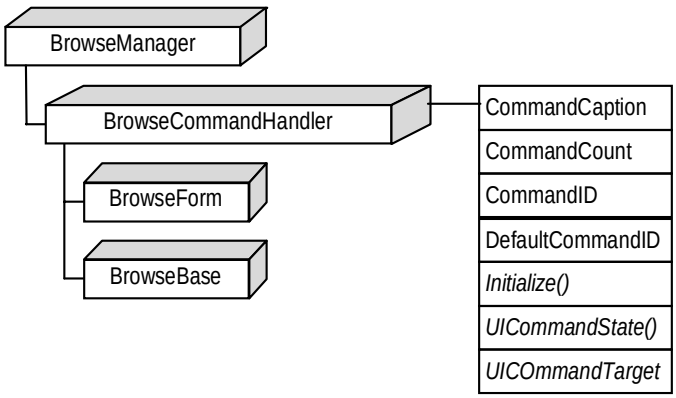
See Also

RangeOption Property, page 60, **SearchKey Property**, page 63, and **UseDistinctKey Property**, page 66.

BrowseCommandHandler Class

This class defines objects that are used by the BrowseManager class to organize and control custom command buttons added to browse dialogs. The browse dialog is notified as to which browse command handler services each custom command.

The following diagram shows the structure of the BrowseCommandHandler class and how it is linked to the BrowseManager class:



Structure of the BrowseCommandHandler Class

The BrowseCommandHandler class has the following methods and properties:

CommandCaption Property, page 69	**Initialize Method**, page 72
CommandCount Property, page 70	**UICommandState Method**, page 72
CommandID Property, page 70	**UICommandTarget Method**, page 73
DefaultCommandID Property, page 71	

CommandCaption Property

This property returns a caption for the command specified by the Index parameter.

Syntax

object.CommandCaption(*Index*)

Part	Required	Data Type	Description
Index	yes	long	Index to the commands handled by the command handler.

Remarks

BrowseManager uses this property to add a command button to a browse form.

Example

```
' Set the command handler's data object to reference the
' BrowseManager's BrowseBase object and the browse form.
cmdHandler.Initialize DataObject, BrowseForm

' For each command handled by the command handler ...
For i = 1 To cmdHandler.CommandCount

  ' Add a command button to the browse form.
  BrowseForm.AddButton cmdHandler.CommandId(i), _
                       cmdHandler, _
                       cmdHandler.CommandCaption(i)
Next i

' Set the browse form's default action.
BrowseForm.DefaultAction = cmdHandler.DefaultCommandId
```

Data Type

String

See Also

CommandID Property, page 70.

CommandCount Property

This property returns a count of the commands handled by the command handler.

Syntax

object.CommandCount

Remarks

When adding command buttons to the browse form, the BrowseManager class uses this property to loop through all the available commands handled by the command handler.

Example

See **CommandCaption Property**, page 69.

Data Type

Long

CommandID Property

This property returns a command ID for the command specified by the Index parameter.

Syntax

object.CommandID(*Index*)

Part	Required	Data Type	Description
Index	yes	long	Index into the commands handled by the command handler.

Remarks

This property uniquely identifies a command within all commands handled by the command handler. To “hook” the command to a toolbar button or menu in the MDI frame, use the same command ID as the multiple-document interface (MDI) frame command.

Example

See **CommandCaption Property**, page 69.

Data Type

Long

DefaultCommandID Property

This property returns a command ID for the default command on a browse dialog.

Syntax

object.DefaultCommandID

Remarks

It is not necessary to set a default command ID.

Example

See **CommandCaption Property**, page 69.

Data Type

Long

Initialize Method

This method initializes the command handler with two references: a reference to the BrowseBase object and a reference to the browse form.

Syntax

```
object.Initialize(BrowseBase, BrowseForm)
```

Part	Required	Data Type	Description
BrowseBase	yes	BrowseBase	Reference to the BrowseBase object that is linked to the browse form.
BrowseForm	yes	Form	Reference to the browse form that issues the commands handled by the command handler.

Remarks

The command handler can use the reference to the browse form (initialized by this method) to update properties on the browse form (for example, GUI controls); the command handler can use the reference to the BrowseBase object to access the object's data cache.

UICommandState Method

This method handles or forwards the commands sent to the specified Browse object from the multiple-document interface (MDI) frame.

Syntax

```
object.UICommandState (Command, ForwardToNext)
```

Part	Required	Data Type	Description
Command	yes	UICmd	User interface command.
ForwardToNext	yes	Boolean	Forward to next command handler.

See Also

UICommandTarget Method, page 73, and **Command Handler Classes**, page 183.

UICommandTarget Method

This method handles the commands sent to the specified Browse object from the multiple-document interface (MDI) frame, if possible, and then forwards the commands to the next command handler linked to the MDI frame.

Syntax

object.UICommandTarget (*Command*, *ForwardToNext*)

Part	Required	Data Type	Description
Command	yes	UICmd	User interface command.
ForwardToNext	yes	Boolean	Forward to next command handler.

Remarks

This method contains a Select statement that includes a case statement for every command handled by the command handler.

See Also

UICommandState Method, page 72, and **Command Handler Classes**, page 183.

BrowseDataCache Class

This class defines objects that store data returned from the server or data pre-loaded during initialization; these objects also store the data format information.

The BrowseDataCache class has the following methods and properties:

Columns Property, page 74	**Rows Property**, page 76
GetValue Method, page 75	

Columns Property

This property defines a collection of objects, each of which maintains descriptive information about a specified data column in the cache.

Syntax

`[cache_object].Columns`

Example

```
With DataObject.Cache.Columns
     ColumnName = .Item(index).Name
End With
```

Data Type

BrowseDataColumns class

See Also

Rows Property, page 76.

GetValue Method

This method returns a value from the cached data in the specified row and column.

Syntax

```
[cache_object].GetValue(column, row_id, _
                        cached_or_selected_flag, _
                        formatted_or_rawdata_flag, _
                        dim1, dim2)
```

Part	Required	Data Type	Description
column	yes	string	Name of the column to retrieve the data from.
row_id	yes	variant	Index for the column data.
cached_or _selected_flag	yes	Integer	Indicates whether the Row parameter indexes to cached or selected rows.
formatted_or _rawdata_flag	yes	Integer	Indicates whether retrieved values are formatted as raw data or for display.
dim1	no	variant	Index for the 1st dimension, if the value is an array.
dim2	no	variant	Index for the 2nd dimension, if the value is an array.

Remarks

The value returned may be a single data object or a 1- or 2-dimensional array.

Example

```
With DataObject.Cache
     DataString = .GetValue(ColumnName, _
                            RecordCount, _
                            BR_CACHED_DATA, _
                            BR_FORMATTED_DATA)
End With
```

Data Type

Variant

Rows Property

This property defines a collection of row objects, each of which maintains data for a specified data row in the cache. Each row must have a unique identifier.

Syntax

[*cache_object*].Rows

Example

```
With DataObject.Cache.Rows
     ' Get a row's unique identifier
     Unique_ID = .Item(index).RowID
End With
```

Data Type

BrowseDataRows

See Also

Columns Property, page 74.

BrowseDataColumn Class

This class defines objects that store formatting, array, and display information for a specified data column.

The BrowseDataColumn class has the following methods and properties:

BDT Property

This property gets the business data type (BDT) for a data column.

Syntax

```
format = [data_column_object].BDT
```

Remarks

Use this property to format and validate data.

Example

```
Dim BDT_String as String

With DataObject.Cache.Columns.Item("CUSTOMER")
     BDT_String = .BDT
End With
```

Data Type

String

See Also

- **Caption Property**, page 78
- **Format Property**, page 80
- **HighBound Property**, page 81
- **LowBound Property**, page 82
- **Name Property**, page 83
- **Rank Property**, page 84
- **Visible Property**, page 85

Caption Property

This property gets or sets the caption text for a data column.

Syntax

`[`*`data_column_object`*`].Caption = value`

Part	Required	Data Type	Description
value	yes	string	Caption text for the data column.

Remarks

The caption text is used on the browse dialog and the browse options dialog. This text is independent of the data column label defined in Construct Spectrum.

Example

```
With DataObject.Cache.Columns.Item("CUSTOMER")
     Column_Caption = .Caption
End With
```

Data Type

String

See Also

- **BDT Property**, page 77
- **Format Property**, page 80
- **HighBound Property**, page 81
- **LowBound Property**, page 82
- **Name Property**, page 83
- **Rank Property**, page 84
- **Visible Property**, page 85

Create Method

This method creates a data column definition.

Syntax

```
[data_column_object].Create(column_name, _
                            column_caption, _
                            column_bdt, _
                            column_format, _
                            column_visible, _
                            lowbound1, highbound1, _
                            lowbound2, highbound2)
```

Part	Required	Data Type	Description
column_name	yes	string	Name of the column.
column_caption	yes	string	Caption text for the column.
column_bdt	yes	string	Business data type for the column.
column_format	yes	string	Format for the column.
column_visible	optional	variant	Indicates whether the column is visible (True) or not (False).
lowbound1	optional	variant	Lower bound row index.
highbound1	optional	variant	Upper bound row index.
lowbound2	optional	variant	Lower bound column index.
highbound2	optional	variant	Upper bound column index.

Remarks

This method is called during the initialization of business objects.

Example

```
Dim NewColumn As BrowseDataColumn

Set NewColumn = New BrowseDataColumn
NewColumn.CreateName, _
     Caption, _
     BDT, _
     Format, _
     Visible, _
     LowBound1, _
     HighBound1, _
     LowBound2, _
     HighBound2

BaseObj.Cache.Columns.Add NewColumn
```

Format Property

This property gets the Natural format for a data column.

Syntax

```
format_string = [data_column_object].Format
```

Remarks

The format string is a descriptor matching the format used in the Natural parameter data area (PDA) description of the data column.

Example

```
Dim Format_String as String

With DataObject.Cache.Columns.Item("CUSTOMER")
     Format_String = .Format
End With
```

Data Type

Variant

See Also

- **BDT Property**, page 77
- **Caption Property**, page 78
- **HighBound Property**, page 81
- **LowBound Property**, page 82
- **Name Property**, page 83
- **Rank Property**, page 84
- **Visible Property**, page 85

HighBound Property

This property gets the upper bound value for a data column.

Syntax

```
dimension = [data_column_object].HighBound
```

Remarks

This property applies to array columns only.

Example

```
With DataObject.Cache.Columns.Item("ORDER-LINES")
     Upper_Limit = .HighBound
End With
```

Data Type

Integer

See Also

- **BDT Property**, page 77
- **Caption Property**, page 78
- **Format Property**, page 80
- **LowBound Property**, page 82
- **Name Property**, page 83
- **Rank Property**, page 84
- **Visible Property**, page 85

Justification Property

This property gets or sets the justification format for a data column (left, right, or center).

Syntax

```
[data_column_object].Justification = style
```

Part	Required	Data Type	Description
style	yes	Integer	Justification setting. For example: `CF_LEFT_JUSTIFIED | CF_RIGHT_JUSTIFIED | CF_CENTER`

Remarks

Use this property to format data for display in a column.

Example

```
With DataObject.Cache.Columns.Item("CUSTOMER")
     .Justification = CF_LEFT_JUSTIFIED
End With
```

Data Type

Integer

See Also

Caption Property, page 78, and **Format Property**, page 80.

LowBound Property

This property gets the lower bound value for a data column.

Syntax

```
dimension = [data_column_object].LowBound
```

Remarks

This property applies to array columns only.

Example

```
With DataObject.Cache.Columns.Item("ORDER-LINES")
     Lower_Limit = .LowBound
End With
```

Data Type

Integer

See Also

- **BDT Property**, page 77
- **Caption Property**, page 78
- **Format Property**, page 80
- **HighBound Property**, page 81
- **Name Property**, page 59
- **Rank Property**, page 84
- **Visible Property**, page 85

Name Property

This property gets the name of a data column.

Syntax

```
column_name = [data_column_object].Name
```

Remarks

This name is the same as the data column name defined in the Construct Spectrum parameter data area (PDA) for the Browse object. It is independent of the column caption defined in Construct Spectrum.

Example

```
With DataObject.Cache.Columns.Item("CUSTOMER")
     Column_Name = .Name
End With
```

Data Type

String

See Also

- **BDT Property**, page 77
- **Caption Property**, page 78
- **Format Property**, page 80
- **HighBound Property**, page 81
- **LowBound Property**, page 82
- **Rank Property**, page 84
- **Visible Property**, page 85

Rank Property

This property gets the rank (dimension) of an array value.

Syntax

```
dimensions = [data_column_object].Rank
```

Remarks

This property applies to array columns only.

Example

```
With DataObject.Cache.Columns.Item("CUSTOMER")
     Column_Rank = .Rank
End With
```

Data Type

Integer

See Also

- **BDT Property**, page 77
- **Caption Property**, page 78
- **Format Property**, page 80
- **HighBound Property**, page 81
- **LowBound Property**, page 82
- **Name Property**, page 59
- **Visible Property**, page 85

Visible Property

This property gets or sets the display value for a data column.

Syntax

```
[data_column_object].Visible = {True | False}
```

Remarks

This property defines the visibility of the column as used by the display list for the browse dialog.

Example

```
With DataObject.Cache.Columns.Item("CUSTOMER")
     .Visible = True
End With
```

Data Type

Boolean

See Also

- **BDT Property**, page 77
- **Caption Property**, page 78
- **Format Property**, page 80
- **HighBound Property**, page 81
- **LowBound Property**, page 82
- **Name Property**, page 59
- **Rank Property**, page 84

BrowseDataColumns Class

This class defines a collection of BrowseDataColumn objects and the methods used to manipulate the objects. It has the following methods and properties:

Add Method

This method adds a new column to the internal BrowseDataColumns object collection.

Syntax

[*data_columns_object*].Add(column)

Part	Required	Data Type	Description
column	yes	BrowseDataColumn	Name of the new column.

Remarks

This method is called internally during the initialization of business objects.

Example

```
Dim NewColumn As BrowseDataColumn

Set NewColumn = New BrowseDataColumn
NewColumn.Create  Name, _
                  Caption, _
                  BDT, _
                  Format, _
                  Visible, _
                  LowBound1, _
                  HighBound1, _
                  LowBound2, _
                  HighBound2

BaseObj.Cache.Columns.Add NewColumn
```

See Also

Clear Method, page 87, **Item Method**, page 88, and **Remove Method**, page 88.

Clear Method

This method removes all members from the internal BrowseDataColumns object collection.

Syntax

[*data_columns_object*].Clear

See Also

Add Method, page 86, **Item Method**, page 88, and **Remove Method**, page 88.

Count Property

This method returns a count of all BrowseDataColumns objects in the internal Browse-DataColumns object collection.

Syntax

[*data_columns_object*].Columns

Example

```
With DataObject.Cache.Rows
     Total_Rows = .Count
End With
```

Data Type

Long

Item Method

This method returns a BrowseDataColumn object from the internal BrowseDataColumns object collection.

Syntax

```
Set column_object = [data_columns_object].Item (index)
```

Part	Required	Data Type	Description
index	yes	variant	Column identifier or positional index.

Example

```
Dim DataColumn As BrowseDataColumn

With DataColumns
     ' Get dimension information for the column
     Set DataColumn = .Item("CUSTOMER")
End With
```

Data Type

BrowseDataColumn

See Also

Add Method, page 86, and **Remove Method**, page 88.

Remove Method

This method removes a column definition from the internal BrowseDataColumns object collection.

Syntax

```
[data_columns_object].Remove(index)
```

Part	Required	Data Type	Description
index	yes	variant	Column identifier or positional index.

Example

```
BaseObj.Cache.Columns.Remove("CUSTOMER")
```

See Also

Add Method, page 86, and **Item Method**, page 88.

SetAllVisible Method

This method sets the Visible property for all defined columns.

Syntax

[*data_columns_object*].SetAllVisible state

Part	Required	Data Type	Description
state	yes	Integer (Boolean)	Visible property value for all defined columns. Valid values are True or False.

Remarks

This method sets the Visible property to the same state for all defined columns.

Example

```
BaseObj.Cache.Columns.SetAllVisible True
```

See Also

SetVisible Method, page 90.

SetVisible Method

This method sets the Visible property for a specified column.

Syntax

[*data_columns_object*].SetVisible *column_name*, *state*

Part	Required	Data Type	Description
column_name	yes	string	Name of the column.
state	yes	Integer (Boolean)	Visible property value for the specified column. Valid values are True or False.

Example

```
BaseObj.Cache.Columns.SetVisible "CUSTOMER", True
```

See Also

SetVisible Method, page 90.

BrowseDataRow Class

This class defines objects that contain data for all columns in a data row and information about methods and attributes for data rows.

The BrowseDataRow class has the following methods and properties:

CreateWithData Method, page 91	**Selected Property**, page 93
CreateWithPDA Method, page 92	**State Property**, page 94
RowID Method, page 93	**Value Method**, page 95

CreateWithData Method

This method creates a new BrowseDataRow object using the static data for a Browse object.

Syntax

`[`*`data_row_object`*`].CreateWithData(`*`data_columns`*`, `*`data`*`)`

Part	Required	Data Type	Description
data_columns	yes	BrowseDataColumns	Data columns object describing the contents and format of the data area.
data	yes	ParamArray variant	Matching data array.

Remarks

This method creates a data row by initializing the column values with data passed in the ParamArray variant. The data values must be consistent with the columns specified in the BrowseDataColumns object. Any multi-dimensional columns are assumed to have only one initial entry.

Example

```
Set NewRow = New BrowseDataRow
NewData = Data(0)
NewRow.CreateWithData BaseObj.Cache.Columns, Data()
```

See Also

CreateWithPDA Method, page 92.

CreateWithPDA Method

This method creates a new BrowseDataRow object using the results parameter data area (PDA) for a Browse object.

Syntax

```
[data_row_object].CreateWithPDA(data_columns, rows_pda, row_index)
```

Part	Required	Data Type	Description
data_columns	yes	BrowseDataColumns object	Data columns object describing the contents and format of the data area.
rows_pda	yes	NaturalDataArea object	Internal results area in the business object.
row_index	yes	long	Index for the new row.

Remarks

This method creates a data row by initializing each column's value with data from the business object's query results area. The column names in the specified BrowseData-Columns object must match the field names in the business object.

Example

```
' Create the new row.
Set NewRow = New BrowseDataRow
NewRow.CreateWithPDA Cache.Columns, RowsPDA, RowIndex
NewRow.RowID = UniqueRowID

' Add it to the rows cache.
Cache.Rows.Add NewRow
```

See Also

CreateWithPDA Method, page 92.

RowID Method

This method gets or sets a unique identifier for a specified row.

Syntax

```
value = [data_row_object].RowID
```

Part	Required	Data Type	Description
value	yes	string	Unique identifier.

Remarks

The row ID must uniquely identify the row within the collection of rows. If duplicate identifiers are specified, a runtime error occurs.

Example

```
' Ensure that the unique row ID is a string.
NewRow.RowID = "a" & NewRow.RowID
BaseObj.Cache.Rows.Add NewRow
```

Data Type

String

See Also

Value Method, page 95.

Selected Property

This property gets or sets the Selected value for a data cache row.

Syntax

```
[data_row_object].Selected = {True | False}
```

Remarks

You can adjust the Selected property for a specified row either through direct manipulation or as the result of an action performed by a browse dialog.

Example

```
With DataObject.Cache.Rows.Item(Unique_Id)
     Is_Selected = .Selected
End With
```

Data Type

Boolean

See Also

State Property, page 94.

State Property

This property gets or sets the application-defined state for a row.

Syntax

[*data_row_object*].State = value

Part	Required	Data Type	Description
value	yes	long	Application-defined state for a row.

Remarks

Use this property to assign a state to a row (for example, Deleted, Updated, or Locked). You can also use this property as an index to an image list that places a small icon on the data display for the browse dialog. You can extend or modify the image list.

Example

```
With DataObject.Cache.Rows.Item(Unique_Id)
     Is_Locked = .State
End With
```

Data Type

Long

See Also

Selected Property, page 93.

Value Method

This method returns a value from the DataCells collection.

Syntax

```
column_value = [data_row_object].Value(column_name, dim1, dim2)
```

Part	Required	Data Type	Description
column_name	yes	string	Name of the column.
dim1	no	variant	Row index for 1- or 2-dimensional arrays.
dim2	no	variant	Column index for 2-dimensional arrays.

Remarks

The value may be of any type supported by the business data types (BDTs).

Example

```
With DataObject.Cache.Rows.Item(Unique_Id)
     Column_Value = .Value("CUSTOMER")
End With
```

Data Type

Variant

See Also

RowID Method, page 93.

BrowseDataRows Class

This class defines a collection of BrowseDataRow objects and the methods used to access and manipulate the objects either individually or collectively.

The BrowseDataRows class has the following methods and properties:

Add Method

This method adds a new row to the internal BrowseDataRows object collection.

Syntax

`[`*`data_rows_object`*`].Add(`*`browse_data_row`*`)`

Part	Required	Data Type	Description
browse_data_row	yes	BrowseDataRow	New row for the BrowseDataRows object.

Remarks

This method is called during the initialization of business objects that contain fixed or relatively static lists.

Example

```
Dim NewRow As BrowseDataRow

Set NewRow = New BrowseDataRow
With NewRow
     .CreateWithData Cache.Columns, Data
     .RowID = UniqueRowID
End With

With DataObject.Cache.Rows
     .Add NewRow
End With
```

See Also

SelectedCount Property, page 103.

Clear Method

This method removes all rows from the internal BrowseDataRows object collection.

Syntax

[*data_rows_object*].Clear

Remarks

This method clears and resets the internal BrowseDataRows object collection in the data cache area to prepare for new data.

Example

```
With DataObject.Cache.Rows
     .Clear
End With
```

See Also

Add Method, page 96.

Count Property

This property returns a count of the rows in the internal BrowseDataRows object collection.

Syntax

```
[data_rows_object].Count
```

Example

```
With DataObject.Cache.Rows
     Total_Rows = .Count
End With
```

Data Type

Long

See Also

SelectedCount Property, page 103.

GetSelectedValue Method

This method gets the value of a row and column that are marked as Selected.

Syntax

```
[data_rows_object].GetSelectedValue(row_id, column_name, dim1, dim2)
```

Part	Required	Data Type	Description
row_id	yes	variant	String that uniquely identifies a row or positional index in the internal BrowseDataRows object collection.
column_name	yes	string	Name of the column for which the value is returned.
dim1	optional	variant	If the value is from a 1- or 2-dimensional array, the row index.
dim2	optional	variant	If the value is from a 2-dimensional array, the column index.

Remarks

You can adjust this property either through direct manipulation or as the result of an action performed by a browse dialog.

Example

```
With DataObject.Cache.Rows
     Value = .GetSelectedValue(Unique_Id, "CUSTOMER")
End With
```

Data Type

Variant

Item Method

This method returns a BrowseDataRow object from the internal BrowseDataRows object collection.

Syntax

[*data_rows_object*].Item(*row_id*)

Part	Required	Data Type	Description
row_id	yes	variant	String that uniquely identifies a row or positional index in the internal BrowseDataRows object collection.

Remarks

This method simplifies the manipulation of items in the data cache by allowing direct access to row information for the data cache.

Example

```
With DataObject.Cache.Rows
     Value = .Item(Unique_Id).Value("CUSTOMER")
End With
```

Data Type

BrowseDataRow

See Also

SelectedItem Method, page 104.

Remove Method

This method removes a data row from the internal selected BrowseDataRows object collection.

Syntax

[*data_rows_object*].Remove(*row_id*)

Part	Required	Data Type	Description
row_id	yes	variant	String that uniquely identifies a row or positional index in the internal BrowseDataRows object collection.

Example

```
With DataObject.Cache.Rows
     .Remove(Unique_Id)
End With
```

See Also

RemoveAllSelectedRows Method, page 100, and **RemoveSelectedRow Method**, page 101.

RemoveAllSelectedRows Method

This method removes all selected rows from the internal selected BrowseDataRows object collection.

Syntax

[*data_rows_object*].RemoveAllSelectedRows

Remarks

You can adjust the Selected property either through direct manipulation or as the result of an action performed by a browse dialog.

Example

```
With DataObject.Cache.Rows
     .RemoveAllSelectedRows
End With
```

See Also

RemoveSelectedRow Method, page 101.

RemoveSelectedRow Method

This method removes the selected data row from the internal selected BrowseDataRows object collection.

Syntax

[*data_rows_object*].RemoveSelectedRow(*row_id*)

Part	Required	Data Type	Description
row_id	yes	variant	String that uniquely identifies the selected row or positional index in the internal BrowseDataRows object collection.

Remarks

You can adjust the Selected property either through direct manipulation or as the result of an action performed by a browse dialog.

Example

```
With DataObject.Cache.Rows
     .RemoveSelectedRow Index
End With
```

See Also

RemoveAllSelectedRows Method, page 100.

SelectAllRows Method

This method selects all rows in the data cache.

Syntax

```
[data_rows_object].SelectAllRows
```

Remarks

This method defines the Selected property for each row as True and adds the rows to the internal selected BrowseDataRows object collection.

Example

```
With DataObject.Cache.Rows
     .SelectAllRows
End With
```

See Also

UnSelectAllRows Method, page 105.

Selected Property

This property gets or sets the Selected value for a data cache row.

Syntax

```
[data_rows_object].Selected(row_id) = value
```

Part	Required	Data Type	Description
row_id	yes	variant	String that uniquely identifies the selected row or positional index in the internal selected BrowseDataRows object collection.
value	yes	Integer (Boolean)	Selected property value for the data cache row. Valid values are True or False.

Remarks

You can adjust this property either through direct manipulation or as the result of an action performed by a browse dialog.

Example

```
With DataObject.Cache.Rows
     Is_Selected = .Selected(Unique_Id)
End With
```

Data Type

Boolean

See Also

SelectedCount Property, page 103.

SelectedCount Property

This property returns a count of the rows that are marked as Selected.

Syntax

[*data_rows_object*].SelectedCount

Remarks

You can adjust this property either through direct manipulation or as the result of an action performed by a browse dialog.

Example

```
With DataObject.Cache.Rows
     Total_Selected = .SelectedCount
End With
```

Data Type

Long

See Also

Selected Property, page 102, and **Count Property**, page 98.

SelectedItem Method

This method returns the specified BrowseDataRow object from the internal selected BrowseDataRows object collection.

Syntax

[*data_rows_object*].SelectedItem(*row_id*)

Part	Required	Data Type	Description
row_id	yes	variant	String that uniquely identifies the selected row or positional index in the internal BrowseDataRows object collection.

Remarks

The internal collection of selected BrowseDataRows objects is a subset of the total data cache row collection. Use this method when processing the currently selected row members of the cache. As with other selection methods, you can adjust the Selected property either through direct manipulation or as the result of an action performed by a browse dialog.

Example

```
' Return the value as raw data.
With DataObject.Cache.Rows
     GetValue = .SelectedItem(RowID).Value(Column, Dim1, Dim2)
End With
```

Data Type

BrowseDataRow class

See Also

- **Item Method**, page 88
- **RemoveAllSelectedRows Method**, page 100
- **RemoveSelectedRow Method**, page 101
- **SelectAllRows Method**, page 102
- **UnSelectAllRows Method**, page 105

UnSelectAllRows Method

This method marks the current selections as unselected.

Syntax

[*data_rows_object*].UnSelectAllRows

Remarks

You can adjust the Selected property either through direct manipulation or as the result of an action performed by a browse dialog.

Example

```
With DataObject.Cache.Rows
     .UnSelectAllRows
End With
```

See Also

RemoveAllSelectedRows Method, page 100, **RemoveSelectedRow Method**, page 101, and **SelectAllRows Method**, page 102.

BrowseDialogBase Class

This class allows you to programmatically perform most actions users manually perform. There are two types of browse dialog: multiple-document interface (MDI) and non-MDI. Any differences between these types (other than rules regarding the use of modal vs. non-modal dialogs) are noted in the applicable Remarks section.

Methods and properties for the BrowseDialog class can be grouped as follows:

- **Display Setup**, page 106
  Controls the appearance of the browse dialog.
- **Data Fetch Setup**, page 131
  Controls the search values and keys used to get data from the business objects.
- **Selection List Results**, page 141
  Obtains the results for user selections.
- **Miscellaneous**, page 142
  - Controls a subset of the options a user may later modify

    or
  - Accesses items or methods not included in the above

Display Setup

To control the appearance of the browse dialog, the BrowseDialogBase class has the following methods and properties:

AddButton Method, page 107	**GetLogicalKeyVisibility Method**, page 118
BrowserHeight Property, page 108	**GetPanelVisibility Method**, page 119
BrowserLeft Property, page 109	**MultiSelect Property**, page 120
BrowserTop Property, page 109	**RestoreKeys Method**, page 120
BrowserWidth Property, page 110	**ReSync Method**, page 121
BuildColumnsDisplay Method, page 110	**ReSyncData Method**, page 121
ButtonCount Property, page 111	**SetColumnVisibility Method**, page 122
DataRows Property, page 112	**SetKeyEnabled Method**, page 123
DeleteButton Method, page 112	**SetKeyLock Method**, page 124

AddButton Method

This method adds a button to a browse dialog. When this button is pressed, code in an application-specified command handler object is executed to support an application-defined function.

Syntax

`[`*`form_object`*`].AddButton` *`NewTag, TagHandler, Caption`*

Part	Required	Data Type	Description
NewTag	yes	string	Unique identifier for the new button on the browse dialog.
TagHandler	yes	object	Object with an interface to which the browse dialog passes control after a Click event for a button.
Caption	yes	string	Text displayed on the face of the button.

Remarks

Typically, the Browse object and browse dialog are generated by the Construct Spectrum browse manager. The command handler, however, is almost always a custom class designed by the user. This class must support the two methods required to register and receive commands from the browse dialog. (The browse dialog rejects attempts to add a button with the same identifier.)

Example

The following example assumes a previously defined command handling object, cmdHandler, which has the CommandId and CommandCaption string arrays as internal properties:

```
' Add a button for each active command handled by the command handler.
With BrowseForm
     For i = 1 To cmdHandler.CommandCount
          .AddButton cmdHandler.CommandId(i),
          cmdHandler,
          cmdHandler.CommandCaption(i)
     Next I
End With
```

See Also

AppButton Method, page 142, **DeleteButton Method**, page 112, and **ButtonCount Property**, page 111.

BrowserHeight Property

This property sets or gets the height of the browse dialog in the currently selected coordinate system.

Syntax

[*form_object*].BrowserHeight = [height]

Remarks

This property determines the height of the browse dialog. A minimum value is enforced; the value depends on the visible options on the dialog and on the screen resolution.

Data Type

Integer

See Also

- **BrowserLeft Property**, page 109
- **BrowserTop Property**, page 109
- **BrowserWidth Property**, page 110
- **DataRows Property**, page 112

BrowserLeft Property

This property sets or gets the left side coordinate for the browse dialog in the currently selected coordinate system.

Syntax

```
[form_object].BrowserLeft = [leftside]
```

Remarks

This property determines the starting point for the left edge of the browse dialog. A minimum and maximum value is enforced; the value depends on the visible options on the dialog and on the screen resolution.

Data Type

Integer

See Also

- **BrowserHeight Property**, page 108
- **BrowserTop Property**, page 109
- **BrowserWidth Property**, page 110
- **DataRows Property**, page 112

BrowserTop Property

This property sets or gets the top edge coordinate for the browse dialog in the currently selected coordinate system.

Syntax

```
[form_object].BrowserTop = [topedge]
```

Remarks

This property determines the top edge starting point for the browse dialog form. A minimum and maximum value is enforced; the value depends on the visible options on the dialog and on the screen resolution.

Data Type

Integer

See Also

- **BrowserHeight Property**, page 108
- **BrowserLeft Property**, page 109
- **BrowserWidth Property**, page 110
- **DataRows Property**, page 112

BrowserWidth Property

This property sets or gets the width of the browse dialog in the currently selected coordinate system.

Syntax

```
[form_object].BrowserWidth = [width]
```

Remarks

This property determines the width of the browse dialog. A minimum and maximum value is enforced; the value depends on the visible options on the dialog and on the screen resolution.

Data Type

Integer

See Also

- **BrowserHeight Property**, page 108
- **BrowserLeft Property**, page 109
- **BrowserTop Property**, page 109
- **DataRows Property**, page 112

BuildColumnsDisplay Method

This method redraws the data list according to the specified view state.

Syntax

```
[form_object].BuildColumnsDisplay(view_state)
```

Part	Required	Data Type	Description
view_state	yes	variant	If True, all fields are displayed; if False, only key fields are displayed.

Remarks

Use this method to redraw the data list. Current column visibility settings are used and the ViewState property is set to the value specified in the argument for the method.

Example

```
' Trigger a data list redraw, without changing the current view state
With BrowseForm
     .BuildColumnsDisplay .ViewState
End With
```

See Also

ReSyncData Method, page 121, and **ViewState Property**, page 129.

ButtonCount Property

This property returns a count of the application-defined buttons on a browse dialog.

Syntax

```
[count] = [form_object].ButtonCount
```

Remarks

This property returns the total number of dialog buttons — minus the number of pre-defined buttons.

Data Type

Integer

See Also

- **AddButton Method**, page 107
- **AppButton Method**, page 142
- **DefaultAction Property**, page 143
- **DeleteButton Method**, page 112

DataRows Property

This property sets the minimum number of data rows displayed on the data list for a browse dialog.

Syntax

```
[form_object].DataRows = [rows]
```

Remarks

This property setting interacts with the BrowserHeight property setting to determine the minimum displayed height of the browse dialog.

Data Type

Integer

See Also

- **BrowserHeight Property**, page 108
- **BrowserLeft Property**, page 109
- **BrowserTop Property**, page 109
- **BrowserWidth Property**, page 110

DeleteButton Method

This method deletes an application-defined browse dialog button.

Syntax

```
[form_object].DeleteButton Tag
```

Part	Required	Data Type	Description
Tag	yes	string	Unique identifier for the browse dialog button.

Remarks

The browse dialog rejects attempts to delete non-existent or pre-defined buttons.

Example

```
' Delete each button previously added for the command handler.
With BrowseForm
     For i = 1 To cmdHandler.CommandCount
          .DeleteButton cmdHandler.CommandId(i)
     Next I
End With
```

See Also

- **AddButton Method**, page 107
- **AppButton Method**, page 142
- **DefaultAction Property**, page 143
- **ButtonCount Property**, page 111

DisableKey Method

This method removes a specified key from the selection key combobox.

Syntax

[*form_object*].DisableKey *KeyName*

Part	Required	Data Type	Description
KeyName	yes	string	Name of the logical key.

Remarks

This method removes a logical key from the logical key list for a browse dialog. The key may be restored by re-synchronizing the dialog and business object (using the RestoreKeys or Resync methods).

Example

```
' Disable (hide) a logical key.
With BrowseForm
     .DisableKey "CUSTOMER-ID"
End With
```

See Also

DisableKeys Method, page 114, and **RestoreKeys Method**, page 120.

DisableKeys Method

This method removes all keys from the logical key selection combobox and all key text boxes from the browse dialog.

Syntax

```
[form_object].DisableKeys
```

Example

```
' Disable (hide) all logical keys.
With BrowseForm
     .DisableKeys
End With
```

See Also

DisableKey Method, page 113, and **RestoreKeys Method**, page 120.

Form_Resize Method

This method reacts to or emulates a Resize event by moving and resizing display elements as necessary.

Syntax

```
[form_object].Form_Resize
```

Remarks

After an application performs a significant amount of fine-tuning on a browse dialog's appearance, it may be necessary to move or resize display elements to keep them visible and accessible. This method handles the required display changes; it is used internally when the user resizes the dialog (or for any other event that affects the size of the dialog).

Example

```
' Rearrange the browse dialog's buttons, fields and data list.
With BrowseForm
     .Form_Resize
End With
```

See Also

Form_Deactivate Method, page 144.

GetColumnVisibility Method

This method returns the Visibility value for a specified data column on a browse dialog.

Syntax

```
Visibility = [form_object].GetColumnVisibility(ColumnName)
```

Part	Required	Data Type	Description
ColumnName	yes	string	Name of the data column.

Remarks

This Boolean value is True when the data column is visible on the data list for the browse dialog.

Example

```
' If the data column is visible, blank the corresponding key field.
With BrowseForm
     If .GetColumnVisibility ("Customer") Then
          .SetKeyValue "Customer", ""
     End If
End With
```

Data Type

Integer (Boolean)

See Also

- **GetKeyVisibility Method**, page 117
- **GetLogicalKeyVisibility Method**, page 118
- **GetPanelVisibility Method**, page 119
- **SetColumnVisibility Method**, page 122
- **SetKeyRangeVisibility Method**, page 125
- **SetKeyVisibility Method**, page 126
- **SetLogicalKeyVisibility Method**, page 127
- **SetPanelVisibility Method**, page 128

GetKeyEnabled Method

This method returns the Enabled state for a specified key field on a browse dialog.

Syntax

```
state  = [form_object].GetKeyEnabled FieldName
```

Part	Required	Data Type	Description
FieldName	yes	string	Name of the key field.

Remarks

If the key field is enabled (True), the field is also visible and the user can enter and change text in the field. If a field is not visible, it is also not enabled.

Example

```
' Show the data column if the key field is enabled
With BrowseForm
     If .GetKeyEnabled("Customers") Then
          .SetColumnVisibility "Customers", True
     End If
End With
```

Data Type

Integer (Boolean)

See Also

SetKeyEnabled Method, page 123.

GetKeyLock Method

This method returns the Locked state for a specified key field on a browse dialog.

Syntax

```
state  = [form_object].GetKeyLock FieldName
```

Part	Required	Data Type	Description
FieldName	yes	string	Name of the key field.

Remarks

The business object may define requirement restrictions for key fields used for its logical keys. A locked key field is one that is mandatory for a specified logical key.

Example

```
' Hide the field if not required by the logical key
With BrowseForm
     If Not .GetKeyLock("Customers") Then
          .SetKeyVisibility "Customers", False
     End If
End With
```

Data Type

Integer (Boolean)

See Also

SetKeyLock Method, page 124.

GetKeyVisibility Method

This method returns the Visibility value for a specified key field on a browse dialog.

Syntax

`state = [`*`form_object`*`].GetKeyVisibility `*`FieldName`*

Part	Required	Data Type	Description
FieldName	yes	string	Name of the key field.

Remarks

If the state for a key field is visible (True), it may be either enabled or disabled. When not visible, it is not enabled.

Example

```
' Show the data column if the key field is visible
With BrowseForm
     If .GetKeyVisibility "Customers" Then
          .SetColumnVisibility "Customers", True
     End If
End With
```

Data Type

Integer (Boolean)

See Also

- **GetLogicalKeyVisibility Method**, page 118
- **GetPanelVisibility Method**, page 119
- **SetColumnVisibility Method**, page 122
- **SetKeyRangeVisibility Method**, page 125
- **SetKeyVisibility Method**, page 126
- **SetLogicalKeyVisibility Method**, page 127
- **SetPanelVisibility Method**, page 128

GetLogicalKeyVisibility Method

This method returns the Visibility value for a logical selection key combobox on a browse dialog.

Syntax

`state = [`*`form_object`*`].GetLogicalKeyVisibility`

Part	Required	Data Type	Description
state	yes	Integer (Boolean)	Visibility value. Valid values are True or False.

Example

```
' Hide the logical key combobox
With BrowseForm
     If Not .GetLogicalKeyVisibility Then
          .SetLogicalKeyVisibility, True
     End If
End With
```

See Also

- **GetKeyVisibility Method**, page 117
- **GetPanelVisibility Method**, page 119
- **SetColumnVisibility Method**, page 122
- **SetKeyRangeVisibility Method**, page 125
- **SetKeyVisibility Method**, page 126
- **SetLogicalKeyVisibility Method**, page 127
- **SetPanelVisibility Method**, page 128

GetPanelVisibility Method

This method returns the Visibility value for a specified status bar panel.

Syntax

`[`*`form_object`*`].GetPanelVisibility` *`panelindex`*

Part	Required	Data Type	Description
panelindex	yes	Integer	Index for status bar panel (leftmost = 1, rightmost = 3).

See Also

- **GetKeyVisibility Method**, page 117
- **GetLogicalKeyVisibility Method**, page 118
- **SetColumnVisibility Method**, page 122
- **SetKeyRangeVisibility Method**, page 125
- **SetKeyVisibility Method**, page 126
- **SetLogicalKeyVisibility Method**, page 127
- **SetPanelVisibility Method**, page 128

MultiSelect Property

This property sets or gets the selection setting for a data list.

Syntax

```
[form_object].MultiSelect = [{True | False}]
```

Setting	Description
True	Extended, multiple selection is allowed.
False	Multiple selection is not allowed.

Remarks

To extend the selection from the previously selected item to the current item, users press Shift and click the mouse or one of the arrow keys. To select or deselect an item on the list, users press Ctrl and click the mouse.

Example

```
' Allow multiple/extended list selection.
With BrowseForm
     .MultiSelect = True
End With
```

RestoreKeys Method

This method restores the selection keys combobox list using the definitions for the Visual Basic business object.

Syntax

```
[form_object].RestoreKeys
```

Remarks

Use this method when an application has previously restricted the use of certain keys using the DisableKey or DisableKeys method. All logical keys are restored.

Example

```
' Restore keys after security check
With BrowseForm
     .RestoreKeys
End With
```

See Also

DisableKey Method, page 113, and **DisableKeys Method**, page 114.

ReSync Method

This method reloads state information from the business object to re-synchronize the display elements (for example, the view state, search keys, or data list).

Syntax

[*form_object*].ReSync

Remarks

Use this method when the business object is directly manipulated by the application in ways that may affect the display (changing the logical key, for example) after the business object has been attached to the dialog using a Set BrowseBase command.

Example

```
' Resync the browse dialog with the business object
' after changing the object’s properties
With BrowseForm
     .ReSync
End With
```

See Also

RestoreKeys Method, page 120, and **ReSyncData Method**, page 121.

ReSyncData Method

This method reloads data from a Browse object and refreshes the data list.

Syntax

[*form_object*].ReSyncData

Remarks

Use this method when the business object is directly manipulated by the application in ways that may affect the data display (performing Fetch commands, for example):

- After the business object has been attached to the dialog using a Set BrowseBase command.

  or

- When an object has data stored in its cache prior to being attached to a browse dialog.

  You must also call this method to refresh the data list when the application changes the Visibility values for a column.

Example

```
' Resync the browse dialog data with the
' business object's data cache
With BrowseForm
     .ReSyncData
End With
```

See Also

RestoreKeys Method, page 120, and **ReSync Method**, page 121.

SetColumnVisibility Method

This method sets the Visibility value for a specified data column.

Syntax

[*form_object*].SetColumnVisibility *ColumnName*, *Value*

Part	Required	Data Type	Description
ColumnName	yes	string	Name of the data column.
Value	yes	Integer (Boolean)	Visibility value. Valid values are True or False.

Remarks

After all desired column Visibility values are changed, call the BuildColumnsDisplay and ReSyncData methods to refresh the values displayed on the data list.

Example

```
' Show the data column if the key field is enabled
With BrowseForm
     If .GetKeyEnabled("Customers") Then
          .SetColumnVisibility "Customers", True
     End If
End With
```

See Also

- **BuildColumnsDisplay Method**, page 110
- **GetColumnVisibility Method**, page 115
- **GetKeyVisibility Method**, page 117
- **GetLogicalKeyVisibility Method**, page 118
- **ReSyncData Method**, page 121
- **SetKeyVisibility Method**, page 126
- **SetKeyRangeVisibility Method**, page 125
- **SetLogicalKeyVisibility Method**, page 127
- **SetPanelVisibility Method**, page 128
- **ViewState Property**, page 129

SetKeyEnabled Method

This method sets the Enabled state (ability to be edited) for a specified key field on a browse dialog.

Syntax

[*form_object*].SetKeyEnabled *FieldName*, *Value*

Part	Required	Data Type	Description
FieldName	yes	string	Name of the key field.
Value	yes	Integer (Boolean)	Enabled value. Valid values are True or False.

Remarks

If the key field is enabled (True), the user can enter and change text in the field. If the field is enabled, it is also visible.

Example

```
' If the Customer key field is enabled, enable the Contact field.
With BrowseForm
     If .GetKeyEnabled "Customer" Then
          .SetKeyEnabled "Contact", True
     End If
End With
```

See Also

GetKeyEnabled Method, page 116.

SetKeyLock Method

This method sets the Locked state for a specified key field on a browse dialog.

Syntax

[*form_object*].SetKeyLock *FieldName*, *Value*

Part	Required	Data Type	Description
FieldName	yes	string	Name of the key field.
Value	yes	Integer (Boolean)	Locked value. Valid values are True or False.

Remarks

The business object may define requirement restrictions for key fields used for its logical keys. A locked key field is one that is mandatory for the specified logical key.

Example

```
' Require the field for the logical key if the column is visible
With BrowseForm
     If .GetColumnVisibility "Customers" Then
          .SetKeyLock("Customers"), True
     End If
End With
```

See Also

GetKeyLock Method, page 116.

SetKeyRangeVisibility Method

This method sets the Visibility value for a range selection combobox.

Syntax

```
[form_object].SetKeyRangeVisibility Value
```

Part	Required	Data Type	Description
Value	yes	Integer (Boolean)	Visibility value. Valid values are True or False.

Remarks

This method shows or hides the key range combobox. You can change the settings using the Options subdialog's copy of the combobox.

Example

```
' Hide the key range combobox
With BrowseForm
     .SetKeyRangeVisibility True
End With
```

See Also

- **GetColumnVisibility Method**, page 115
- **GetKeyVisibility Method**, page 117
- **GetLogicalKeyVisibility Method**, page 118
- **GetPanelVisibility Method**, page 119
- **SetColumnVisibility Method**, page 122
- **SetKeyVisibility Method**, page 126
- **SetLogicalKeyVisibility Method**, page 127
- **SetPanelVisibility Method**, page 128

SetKeyVisibility Method

This method sets the Visibility value for the key field on a browse dialog.

Syntax

```
[form_object].SetKeyVisibility FieldName, Value
```

Part	Required	Data Type	Description
FieldName	yes	string	Name of key field.
Value	yes	Integer (Boolean)	Visibility value. Valid values are True or False.

Remarks

If the field is not visible, it is also not enabled.

Example

```
' Require the field for the logical key if the column is visible
With BrowseForm
     If .GetColumnVisibility "Customers" Then
          .SetKeyVisibility("Customers"), True
     End If
End With
```

See Also

- **GetColumnVisibility Method**, page 115
- **GetKeyVisibility Method**, page 117
- **GetLogicalKeyVisibility Method**, page 118
- **GetPanelVisibility Method**, page 119
- **SetColumnVisibility Method**, page 122
- **SetLogicalKeyVisibility Method**, page 127
- **SetPanelVisibility Method**, page 128

SetLogicalKeyVisibility Method

This method sets the Visibility value for the logical selection key combobox on a browse dialog.

Syntax

```
[form_object].SetLogicalKeyVisibility value
```

Part	Required	Data Type	Description
value	yes	Integer (Boolean)	Visibility value. Valid values are True or False.

Remarks

This method shows or hides the logical key combobox. You can change the settings using the Options subdialog's copy of the combobox.

Example

```
' Hide the logical key combobox
With BrowseForm
     .SetLogicalKeyVisibility False
End With
```

See Also

- **GetColumnVisibility Method**, page 115
- **GetKeyVisibility Method**, page 117
- **GetLogicalKeyVisibility Method**, page 118
- **GetPanelVisibility Method**, page 119
- **SetColumnVisibility Method**, page 122
- **SetKeyVisibility Method**, page 126
- **SetPanelVisibility Method**, page 128

SetPanelVisibility Method

This method sets the Visibility value for a specified status bar panel.

Syntax

```
[form_object].SetPanelVisibility panelindex, visibility
```

Part	Required	Data Type	Description
panelindex	yes	Integer	Index for the status bar panel (leftmost = 1, rightmost = 3).
visibility	yes	Boolean	Visibility value. Valid values are True or False.

Remarks

Hiding a status bar panel does not affect the operation of the browse dialog in any other way. The panel index is rejected if it is out of range.

Example

```
' Hide the middle status bar panel
With BrowseForm
     .SetPanelVisibility 2, False
End With
```

See Also

- **GetColumnVisibility Method**, page 115
- **GetKeyVisibility Method**, page 117
- **GetLogicalKeyVisibility Method**, page 118
- **GetPanelVisibility Method**, page 119
- **SetColumnVisibility Method**, page 122
- **SetKeyVisibility Method**, page 126
- **SetLogicalKeyVisibility Method**, page 127

ShowModal Method

This method shows a browse dialog as a modal dialog and returns success or failure as a Boolean value (True or False).

Syntax

```
[status] = [form_object].ShowModal Mode
```

Part	Required	Data Type	Description
Mode	yes	Integer (Boolean)	Visibility value. Valid values are True or False.

Remarks

You cannot use this method with a multiple-document interface (MDI) browse dialog.

Example

```
' Show a modal browse dialog (i.e., for a static list)
With BrowseForm
     .ShowModal True
End With
```

Data Type

Integer (Boolean)

ViewState Property

This property gets or sets the View value for a data list on a browse dialog.

Syntax

```
[form_object].ViewState = [{True | False}]
```

Setting	Description
True	All columns defined as visible are displayed.
False	Only columns defined as visible key fields are displayed.

Remarks

This property mirrors the All Fields or Keys Only state for a logical key. When the key does not support the Keys Only state, the state defaults to True (all columns are displayed).

Example

```
' Set the data list to "Keys only" mode (COUNT and key fields only).
With BrowseForm
     .ViewState = False
End With
```

See Also

BuildColumnsDisplay Method, page 110, **ReSyncData Method**, page 121, and **SetColumnVisibility Method**, page 122.

Write_StatusBar Method

This method puts text on the status bar panels.

Syntax

[*form_object*].Write_StatusBar(*Message1*, *Message2*, *Message3*)

Part	Required	Data Type	Description
Message1	optional	variant	Text displayed on left status bar panel.
Message2	optional	variant	Text displayed on middle status bar panel.
Message3	optional	variant	Text displayed on right status bar panel.

Remarks

Text defined for the status bar panels should be considered volatile. It may be over-written in the normal course of events.

Example

```
' Write text to the middle panel of the status bar
' and clear the last panel
With BrowseForm
     .Write_StatusBar ,DataString, ""
End With
```

See Also

GetPanelVisibility Method, page 119, and **SetPanelVisibility Method**, page 128.

Data Fetch Setup

The BrowseDialogBase class has the following methods and properties to control the search values and keys used to get data from the business objects:

BrowseBase Property, page 131	**NewQuery Property**, page 136
Fetch Method, page 132	**RowsToRead Property**, page 137
FetchAfterLoad Property, page 132	**SetFixedCount Method**, page 138
GetFixedCount Method, page 133	**SetKeyRange Method**, page 139
GetKeyRange Method, page 134	**SetKeyValue Method**, page 140
GetKeyValue Method, page 135	**SetLogicalKey Method**, page 140
GetLogicalKey Method, page 136	

BrowseBase Property

This property identifies the business object currently attached to a browse dialog.

Syntax

```
Set [form_object].BrowseBase = [business_object]
```

Remarks

Use this property to maintain a reference to a business object's base properties (created when the object was created). To allow the browse dialog to configure itself, this reference must be set.

Example

```
' Set the Browse dialog to a specific business object.
With BrowseForm
     Set .BrowseBase = DataObject
End With
```

Fetch Method

This method gets the requested data from the server and updates the display to match the data in the Browse object.

Syntax

```
[form_object].Fetch
```

Remarks

This method causes the browse dialog to copy its search key values and flags to the business object, and then instruct the business object to request data from the server. When the business object returns, its data cache is copied to the data list on the dialog.

Example

```
' Get data from the server for the display.
With BrowseForm
     .Fetch
End With
```

See Also

NewQuery Property, page 136, and **RowsToRead Property**, page 137.

FetchAfterLoad Property

This property gets or sets the browse dialog flag to perform a data fetch after loading itself.

Syntax

```
[form_object].FetchAfterLoad = [{True | False}]
```

Setting	Description
False	No operation required.
True	Perform a data fetch after opening the dialog, using the current settings of the browse dialog and business object.

Remarks

You can set this property:

- When the application presets the first search values for the browse dialog

  or

- When the business object has search keys set before it is attached to the browse dialog

Example

```
' Set the Browse dialog up to fetch immediately on startup.
With BrowseForm
     .FetchAfterLoad = True
End With
```

Data Type

Integer (Boolean)

See Also

Fetch Method, page 132, and **NewQuery Property**, page 136.

GetFixedCount Method

This method returns a count of the fixed keys defined for a selection key.

Syntax

```
count = [form_object].GetFixedCount
```

Remarks

A logical key may define a number of fields as being fixed with respect to the search results. For example, a key field for “Departments” defined as being fixed with a value of “Admin” only returns results when the value matches exactly. The fields are fixed in order of definition in the key, from most primary key to least important.

If the browse dialog and business object have not been properly synchronized, this method returns –1 (for example, after directly manipulating key definitions for the business object).

Example

```
' Get the fixed key count before attempting to hide fields.
With BrowseForm
     KeyCount = .GetFixedCount
End With
```

Data Type

Integer

See Also

SetFixedCount Method, page 138.

GetKeyRange Method

This method returns the range for a selection key.

Syntax

```
count = [form_object].GetKeyRange
```

Remarks

Displayed Text	Description	Constant
*	Embedded wildcard	BR_EMBEDDED_WILDCARD_RO
<	Less than	BR_LESS_THAN_RO
<=	Less than or equal	BR_LESS_OR_EQUAL_RO
=	Equal	BR_EQUAL_RO
>=	Greater than or equal	BR_GREATER_OR_EQUAL_RO
>	Greater than	BR_GREATER_RO
ABC	Begins with	BR_BEGINS_WITH_RO
blank	No wildcard	BR_NO_WILDCARD_RO

Example

```
' Get the current logical key’s fixed key count.
With BrowseForm
     CurrentRange = .GetKeyRange
End With
```

Data Type

Integer

See Also

SetKeyRange Method, page 139.

GetKeyValue Method

This method returns the value for a specified key field.

Syntax

```
string = [form_object].GetKeyValue FieldName
```

Part	Required	Data Type	Description
FieldName	yes	string	Name of the key field.

Remarks

Use this method to read the text for a key value. The text is formatted according to the business data type (BDT) defined for the key field.

Example

```
' Get the user's "customer" key specification.
With BrowseForm
     ValueString = .GetKeyValue "Customer"
End With
```

Data Type

String

See Also

SetKeyRange Method, page 139.

GetLogicalKey Method

This method returns the current logical key for a browse dialog.

Syntax

```
KeyName = [form_object].GetLogicalKey
```

Remarks

Assuming there is no direct manipulation of the business object, the current logical key setting for the browse dialog reflects that of the business object.

Example

```
' Get the current logical key.
With BrowseForm
     CurrentKey = .GetLogicalKey
End With
```

Data Type

String

See Also

SetLogicalKey Method, page 140.

NewQuery Property

This property gets and sets the New Query flag for a browse dialog.

Syntax

```
[form_object].NewQuery = [{True | False}]
```

Setting	Description
False	Continue using the same logical key and search values.
True	Prepare the browse dialog, the business object, and the server for a new search value.

Remarks

The New Query flag is reflected in the search data area for the business object (as the value of the Restart flag).

Example

```
' Set the browse dialog up to (re)start a search.
With BrowseForm
     .NewQuery = True
End With
```

Data Type

Integer (Boolean)

See Also

FetchAfterLoad Property, page 132.

RowsToRead Property

This property gets and sets a count of the rows read by the business object during one data fetch.

Syntax

[*form_object*].RowsToRead = [number]

Remarks

This variable is reflected in the search data area for the business object.

Example

```
' Set the browse dialog up to fetch 10 records at a time.
With BrowseForm
     .RowsToRead = 10
End With
```

Data Type

Integer

See Also

Fetch Method, page 132, and **NewQuery Property**, page 136.

SetFixedCount Method

This method sets the fixed key count for a selection key.

Syntax

`[form_object].SetFixedCount value`

Part	Required	Data Type	Description
value	yes	Integer	Fixed key count.

Remarks

A logical key may define a number of fields as being fixed with respect to the search results. For example, a key field for “Departments” that is defined as being fixed with a value of “Admin” only returns results when the value matches exactly. The fields are fixed in order of definition in the key, from most primary key to least important.

If necessary, non-visible key fields are considered fixed. This means that a logical key with A, B, C, and D fields cannot have the A and C fields being non-visible without the A through C fields being fixed.

Attempts to set the fixed count to less than the minimum and maximum defined for the logical key in the business object causes the value to be rejected. This does not affect the current setting.

If the browse dialog and business object have not been properly synchronized, this method returns a value of –1 (for example, after directly manipulating the key definitions for the business object).

Example

```
' Set the fixed key count for the current logical key.
With BrowseForm
     .SetFixedCount KeyCount
End With
```

See Also

GetFixedCount Method, page 133.

SetKeyRange Method

This method sets range options for a sort key.

Syntax

[*form_object*].SetKeyRange *value*

Part	Required	Data Type	Description
value	yes	Integer	Index for the key range setting.

Remarks

Displayed Text	Description	Constant
*	Embedded wildcard	BR_EMBEDDED_WILDCARD_RO
<	Less than	BR_LESS_THAN_RO
<=	Less than or equal	BR_LESS_OR_EQUAL_RO
=	Equal	BR_EQUAL_RO
>=	Greater than or equal	BR_GREATER_OR_EQUAL_RO
>	Greater than	BR_GREATER_RO
ABC	Begins with	BR_BEGINS_WITH_RO
blank	No wildcard	BR_NO_WILDCARD_RO

Example

```
' Set the fixed key count to 4 for the current logical key.
With BrowseForm
     .SetKeyRange CONST_GREATER_OR_EQUAL
End With
```

See Also

GetKeyRange Method, page 134.

SetKeyValue Method

This method sets the value for a key field.

Syntax

```
[form_object].SetKeyValue FieldName, Value
```

Part	Required	Data Type	Description
FieldName	yes	string	Name of the key field.
Value	yes	string	Value for the key field.

Remarks

Text for a key field can only be loaded with a string formatted according to the business data type (BDT) defined for the key field.

Example

```
' If the “Customer” data column is visible, blank the key field’s text.
With BrowseForm
     If .GetKeyVisibility("Customer") Then
          .SetKeyValue "Customer", ""
     End If
End With
```

See Also

GetKeyValue Method, page 135.

SetLogicalKey Method

This method sets the logical search key for a browse dialog.

Syntax

```
[form_object].SetLogicalKey KeyName
```

Part	Required	Data Type	Description
KeyName	yes	string	Name of the logical key.

Remarks

This method changes the logical search key that the browse dialog will use on its next data fetch. The setting is propagated to the business object at that time.

Example

```
' Set the logical key to "Personnel by ID code".
With BrowseForm
     .SetLogicalKey "PERSONNEL-ID"
End With
```

See Also

GetKeyValue Method, page 135.

Selection List Results

The BrowseDialog class has the following method to obtain the results for user selections.

CollectSelections Method

This method synchronizes the data selected attributes for a Browse object with the data display attributes.

Syntax

[*form_object*].CollectSelections

Remarks

This method marks the items selected from the data list on the browse dialog as selected in the data cache for the business object. This method is called automatically when the user presses OK to close a browse dialog.

Example

```
' Mark the current data list selections.
With BrowseForm
     .CollectSelections
End With
```

Miscellaneous

The BrowseDialogBase class uses methods and properties listed in this section to:

- Control a subset of the options a user may later modify.

  or

- Access items or methods not included in the previous categories.

The BrowseDialogBase class has the following methods and properties:

AppButton Method, page 142	**Form_Deactivate Method**, page 144
DefaultAction Property, page 143	**SaveColumnSettings Property**, page 145

AppButton Method

This method returns the ApplicationControl object for an application-defined button on a browse dialog.

Syntax

```
AppControl  = [form_object].AppButton Tag
```

Part	Required	Data Type	Description
Tag	yes	string	Unique button identifier.

Remarks

On a browse dialog, an application-defined button is attached to an ApplicationControl object. The ApplicationControl object maintains a reference to the application-defined command handler object for the button.

Example

```
' Get the action button’s handler object.
Dim AppControl as ApplicationControl
Dim cmdHandler as BrowseCommandHandler

With BrowseForm
     cmdHandler = .AppButton("Delete").Handler
End With
```

Data Type

Object

See Also

- **AddButton Method**, page 107
- **DefaultAction Property**, page 143
- **DeleteButton Method**, page 112
- **ButtonCount Property**, page 111

DefaultAction Property

This property gets or sets the default action performed when the user double-clicks a data list on a browse dialog.

Syntax

```
string = [form_object].DefaultAction
```

Remarks

An application can define objects to handle tasks that require a browse dialog (for example, edit or deletion tasks). When the user presses a specified button on a browse dialog, the application can activate the corresponding ApplicationControl object. For example, the application can activate a button when the user selects a value from the data list on the browse dialog.

Example

```
With BrowseForm
     'Add an action button to the browse dialog.
     .AddButton "Delete", cmdHandler, "Delete Record"

     'Set the "Delete" button as the default action.
     .DefaultAction = "Delete"
End With
```

Data Type

String

See Also

AddButton Method, page 107, **DeleteButton Method**, page 112, and **ButtonCount Property**, page 111.

Form_Deactivate Method

This method deactivates a browse dialog form.

Syntax

[*form_object*].Form_Deactivate

Remarks

Use this method whenever focus is transferred from one dialog to another. An application calls this method automatically, but it can also be used by the developer. This method's main function is to unhook menu commands from the multiple-document interface (MDI) child browse dialog.

Example

```
' Unhook an MDI child dialog from the parent's menu.
With BrowseForm
     .Form_Deactivate
End With
```

See Also

Form_Resize Method, page 114.

SaveColumnSettings Property

This property gets or sets the SaveColumnSettings property for a browse dialog.

Syntax

```
[form_object].SaveColumnSettings = {True | False}
```

Remarks

This property controls whether the browse dialog and the data column widths and visibility settings are saved when the user clicks the OK button to exit the application normally. The default setting is True.

Example

```
' Save the display settings for next time.
With BrowseForm
     .SaveColumnSettings = True
End With
```

Data Type

Integer (Boolean)

BrowseManager Class

This class encapsulates all the handling of browse services in a single class. Application components use instances of this class to request browse services. A BrowseManager class is created when an application uses the global GetBrowser() function.

To enhance the functionality of browse forms created by the BrowseManager class, you can set a command handler object as an optional property.

The BrowseManager class has the following methods and properties:

BrowseBase Property, page 146	**GetAllRows Method**, page 149
BrowseByObjectKey Method, page 147	**GetRow Method**, page 150
Caption Property, page 148	**MDIBrowseForm Method**, page 151
CommandHandler Property, page 148	**ModalBrowseForm Method**, page 152
FetchAfterLoad Property, page 149	

BrowseBase Property

This property defines a fully initialized reference to a BrowseBase object; it is a required property.

Syntax

```
set ref = object.BrowseBase
```

Remarks

A BrowseBase object is created when the object factory creates an instance of a specified Browse object. The specified Browse object creates and initializes a BrowseBase object, which can then be used to set this property.

The BrowseManager class bundles browse functionality into several methods. These methods are only enabled when the BrowseBase property for the BrowseManager class is set to an initialized BrowseBase object.

See Also

BrowseBase Class, page 53.

BrowseByObjectKey Method

This method creates a modal browse form. If a row is selected, this method returns True.

Syntax

object.BrowseByObjectKey(*KeyName, ObjectKey*)

Part	Required	Data Type	Description
KeyName	yes	string	Name of the key used by the Visual Basic maintenance object to maintain an object. The key name must also be defined as a logical key in the BrowseBase object.
ObjectKey	yes	NaturalDataArea	Object containing the key structure used by the Visual Basic maintenance object. You must also define the field(s) in this structure as a search key field(s) and a column(s) in the BrowseBase object. Simple keys have only one search key field, while compound keys have multiple search key fields (for example, an Adabas superdescriptor or a DB2 compound key).

Remarks

The search key for the browse form is set to the key specified in the KeyName parameter.

The search key value(s) for the browse form is set to the value(s) contained in the ObjectKey parameter (NaturalDataArea object), where the object key is the key structure used by a Visual Basic maintenance object.

If a row is selected, the key values in the selected row are assigned to the key values in the ObjectKey parameter.

If a row is not selected or the form is canceled, this method returns False.

Note: The fields in the specified Natural data area for the object key must also exist as columns in the BrowseBase object. This is always the case when you use a Visual Basic maintenance object and its corresponding Browse object as sources for the object key, the Natural data area, and the BrowseBase object.

Example

```
Dim BrMgr As BrowseManager

' Only set to True if a new key value was selected.
BrowseByObjectKey = False

' Move the ObjectData into the KeyData.
MoveByNameKey (MOVE_DATA_TO_KEY)

' Invoke a modal dialog to allow selection of a new object key.
Set BrMgr = GetBrowser(OBJECT_TABLE_NAME)
If Br.BrowseByObjectKey(OBJECT_KEY_NAME, _
                        m_ObjectKey.FieldRef("STRUCTURE")) Then
   If KeyChanged() Then
      MoveByNameKey (MOVE_KEY_TO_DATA)
      InvokeMethod "GET"
      BrowseByObjectKey = True
   End If
End If
```

Caption Property

This property gets or sets the caption text for browse forms created by a browse manager; this string is optional.

Syntax

object.caption = *textstring*

CommandHandler Property

This property references a custom command handler object; this reference is optional.

Remarks

A command handler object must implement all the public properties and methods described for the BrowseCommandHandler class.

See Also

BrowseCommandHandler Class, page 68.

FetchAfterLoad Property

This property indicates whether a browse dialog performs a Fetch() operation to retrieve data immediately after loading; this Fetch request is optional.

Syntax

```
object.FetchAfterLoad = True
```

Data Type

Boolean

GetAllRows Method

This method returns all rows from the data cache for a BrowseBase object.

- If a BrowseBase object is linked to a remote data source, this method:
  - Clears the data cache for the BrowseBase object.
  - Issues a FetchAll() request to the remote data source to return all the rows from the remote data source.
  - Returns True if all rows are successfully returned from the remote data source.
- If the BrowseBase object contains static data, this method:
  - Returns False if there are no rows in the data cache for the BrowseBase object.

Syntax

```
object.GetAllRows(KeyName)
```

Part	Required	Data Type	Description
KeyName	no	string	Name of a logical key in the BrowseBase object.

Remarks

If a key name is specified and the BrowseBase object is linked to a remote data source, the returned rows are ordered by the key.

Note: If the BrowseBase object is linked to a remote data source, use caution when using this method. Retrieving all of the rows in a remote table can require excessive transmission time.

Example

```
Dim BrMgr As BrowseManager
Dim KeyName As String
Dim Success As Boolean

KeyName = "CUSTOMER-NUMBER"
Set Br = GetBrowser("NCST-CUSTOMER")
Success = Br.GetAllRows(KeyName)
```

See Also

GetRow Method, page 150.

GetRow Method

This method returns a specified row from the data cache for a BrowseBase object.

- If the BrowseBase object is linked to a remote data source, this method:
  - Clears the data cache for the BrowseBase object.
  - Issues a Fetch(1) request to the remote data source to return a row with a search key name that matches the key value.
- If the BrowseBase object contains static data, this method:
  - Searches the data cache for a row with the key name for a column value that matches the key value.

In both cases, this method returns the RowID property for the row if the row is found; this method returns –1 if the row is not found.

Syntax

object.GetRow(*KeyName, KeyValue*)

Part	Required	Data Type	Description
KeyName	yes	string	Name of a logical key, name of the search key field, and the name of a column in the BrowseBase object. The specified key cannot be a compound key.
KeyValue	yes	variant	Value of the search key field.

Remarks

You can use the returned RowID to access the row; for example:

```
Dim Row As New BrowseDataRow
Set Row = BrowseBase.Cache.Rows.Item(RowID)
```

Example

```
Dim BrMgr As BrowseManager
Dim KeyName As String
Dim KeyValue As String
Dim Row As BrowseDataRow

KeyName = "CUSTOMER-NUMBER"
KeyValue = "100006"
Set Br = GetBrowser("NCST-CUSTOMER")
Row = Br.GetRow(KeyName, KeyValue)
```

Data Type

Variant

See Also

GetAllRows Method, page 149, and **ModalBrowseForm Method**, page 152.

MDIBrowseForm Method

This method creates a multiple-document interface (MDI) browse form and, optionally, links a command handler to the form; this method returns a reference to the form.

Syntax

object.MDIBrowseForm( )

Remarks

This method creates an instance of a generic MDI browse form and formats the form to display the columns and keys contained in the BrowseBase object.

If a command handler is linked to the BrowseManager class, you can use the command handler to:

- Add command buttons to the form.
- Activate toolbar buttons and menus in the MDI frame.
- Place commands on context-sensitive menus that are activated by a right mouse click.
- Handle default commands like double-clicking or pressing the Enter key.

Example

```
Dim BrMgr As BrowseManager
Dim Frm As Form

Set Br = GetBrowser("NCST-ORDER-HEADER")

' Create an instance of a custom command handler, and link it
' to the BrowseManager.
Set Br.CommandHandler = New OrderAsBrowseTarget
Set frm = Br.MDIBrowseForm()
```

See Also

ModalBrowseForm Method, page 152.

ModalBrowseForm Method

This method displays a modal browse dialog and allows users to select a single row. Optionally, this method sets the form's logical key value to KeyName or the search key value to KeyValue.

If a row is selected, this method returns the row ID for the selected row; if a row is not selected, it returns –1.

Syntax

object.ModalBrowseForm(KeyName,KeyValue)

Part	Required	Data Type	Description
KeyName	no	variant	Name of a logical key, the search key field, and the name of a column in the BrowseBase object. The key cannot be a compound key.
KeyValue	no	variant	Value for the search key field.

Remarks

If a command handler is linked to the BrowseBase object, any toolbar buttons or menus on the MDI frame will not be active because the form is modal.

If a command handler is linked to the BrowseBase object, you can use the command handler to:

- Place commands on context-sensitive menus that are activated by a right mouse click.
- Handle default commands like double-clicking or pressing the Enter key.

Example

```
Dim BrMgr As BrowseManager
Dim KeyName As String
Dim KeyValue As String
Dim Row As BrowseDataRow

KeyName = "CUSTOMER-NUMBER"
KeyValue = "100006"
Set Br = GetBrowser("NCST-CUSTOMER")
Row = Br.ModalBrowseForm(KeyName, KeyValue)
```

DataValue

Variant

See Also

GetRow Method, page 150, and **MDIBrowseForm Method**, page 151.

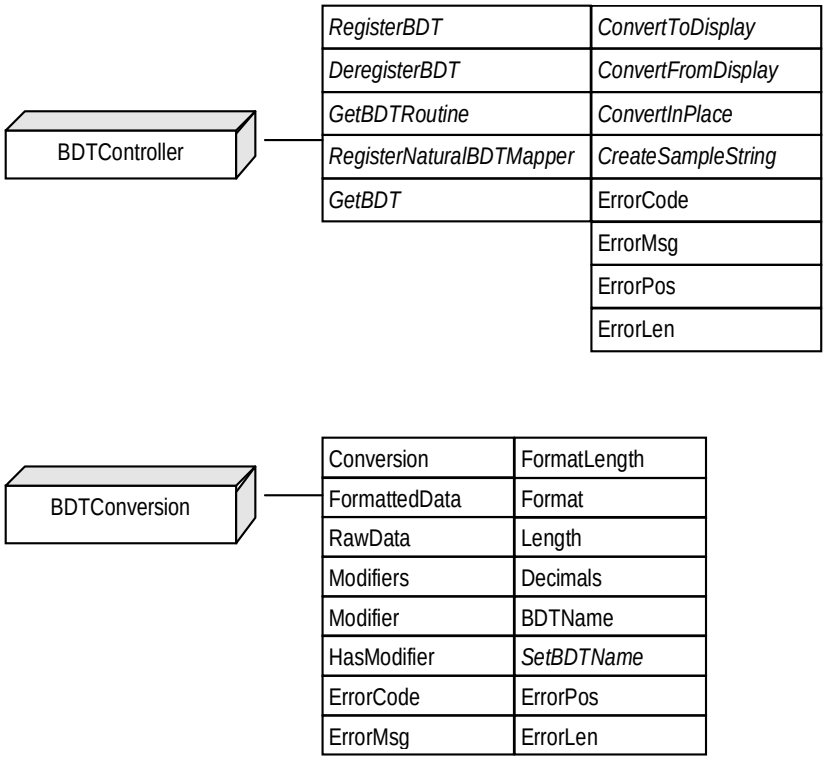
BUSINESS DATA TYPE (BDT) CLASSES

This chapter describes the business data type classes: BDTController and BDTConversion.

The following topics are covered:

Introduction

The client framework includes two classes that provide business data type support: BDTController and BDTConversion. The following diagram shows the structure of the business data type classes:



Structure of the Business Data Type Classes

These methods and properties are described in the following sections.

BDTController Class

The BDTController class registers the business data types (BDTs) used by an application and performs conversions using the BDTs.

Note: The framework declares a global instance of the BDTController object, named BDT, in the Startup.bas module. This variable is used throughout the framework to call the BDT routines.

The properties and methods used by the BDTController class are grouped by function and described in the following sections:

- **Registering BDT Routines**, page 157
- **Calling BDT Routines**, page 162

Registering BDT Routines

To register BDT routines, the BDTController class has the following methods:

DeregisterBDT Method, page 158	**RegisterBDT Method**, page 160
GetBDT Method, page 159	**RegisterNaturalBDTMapper Method**, page 161
GetBDTRoutine Method, page 159	

DeregisterBDT Method

This method deregisters a BDT routine from the BDT controller.

Syntax

object.DeregisterBDT *BDTObject, BDTName*

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object with which BDT routines are registered.
BDTObject	no	object	Reference to object containing the BDT conversion routine.
BDTName	no	string	Name of the BDT to deregister.

Remarks

- If you omit the BDTObject and BDTName parameters, the BDT controller deregisters all BDTs.
- If you include the BDTObject parameter, but omit the BDTName parameter, the BDT controller deregisters all BDTs handled by the object.
- If you include both parameters, the BDT controller deregisters only that BDT conversion routine.
- If you specify the BDTObject parameter, it must be the same instance you specified when registering the BDT; otherwise, the BDTs are not deregistered.

  For example, the following code does not deregister the BDTs:

```
Dim myBDTs As StandardBDTs

Set myBDTs = New StandardBDTs
BDT.RegisterBDT myBDTs, "Convert_Alpha"
BDT.RegisterBDT myBDTs, "Convert_Boolean"

' Create a new instance of the BDTConversion object.
Set myBDTs = New StandardBDTs
BDT.DeregisterBDT myBDTs

' No BDTs are deregistered because a different object is being used.
```

See Also

Registering BDT Routines, page 157.

GetBDT Method

If you omit the BDT name in calls to the conversion routines, this method returns the name (with modifiers) of the BDT used.

Syntax

```
result = object.GetBDT(NatFormatLength)
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object with which the BDT routines are registered.
NatFormatLength	yes	string	Natural format and length.
result		string	Name of the BDT.

Remarks

The BDT name returned is determined by the Natural to BDT mapper routine and set with the RegisterNaturalBDTMapper method.

See Also

GetBDTRoutine Method, page 159.

GetBDTRoutine Method

This method returns the BDTConversion object and the name of the conversion routine for the specified BDT name.

Syntax

```
object.GetBDTRoutine BDTName, Handler, ProcName
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object with which BDT routines are registered.
BDTName	yes	string	Name of the BDT to look up.
Handler	yes	object	Reference to BDTConversion object.
ProcName	yes	string	Name of the conversion routine in the BDTConversion object.

Remarks

The Handler and ProcName parameters are passed by reference. This method supplies the variables.

If the BDT is not registered, the GetBDTRoutine method sets the Handler parameter to Nothing and ProcName parameter to an empty string.

See Also

GetBDT Method, page 159.

RegisterBDT Method

This method registers a BDT. It passes the BDT name, and the names of the object and method that implement the BDT conversion routine, to the BDT controller.

Syntax

```
object.RegisterBDT BDTName, Handler, ProcName
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object with which to register the BDT.
BDTName	yes	string	Name of the BDT to register.
Handler	yes	object	Reference to the BDTConversion object.
ProcName	yes	string	Name of BDT conversion routine in the BDTConversion object.

Remarks

If a BDT with the same name is already registered, the new BDT routine overrides the previous one.

The BDT conversion routine must have the following syntax:

```
Public Sub xxx(BDTC As BDTConversion)
```

where *xxx* is the name of the routine.

See Also

DeregisterBDT Method, page 158.

RegisterNaturalBDTMapper Method

This method registers the Natural to BDT mapper routine.

Syntax

```
object.RegisterNaturalBDTMapper Handler, ProcName
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object with which to register the mapper routine.
Handler	yes	object	Reference to the Natural BDT mapper object.
ProcName	yes	string	Name of the Natural BDT mapper routine in the Handler object.

Remarks

The Natural to BDT mapper routine must have the following syntax:

```
Public Function xxx(Format As String, _
                    Length As Integer, _
                     Decimals As Integer) As String
```

where *xxx* is the name of the routine. This routine must return the BDT name string, which may include modifiers.

See Also

RegisterBDT Method, page 160.

Calling BDT Routines

To call BDT routines, this class has the following methods and properties:

ConvertFromDisplay Method, page 163	**ErrorCode Property**, page 166
ConvertInPlace Method, page 164	**ErrorLen Property**, page 166
ConvertToDisplay Method, page 165	**ErrorMsg Property**, page 167
CreateSampleString Method, page 165	**ErrorPos Property**, page 168

When calling routines, the following information applies to all conversion routines:

- In the call, you must specify: the BDT name, the Natural format and length, or both the BDT name and the Natural format and length.
- If you omit the BDT name, the BDT controller calls the Natural to BDT mapper routine (registered with the RegisterNaturalBDTMapper method) to derive a BDT name from the Natural format and length.
- If you specify both the BDT name and the Natural format and length, the BDT conversion routine may use the Natural format and length to refine the conversion process. For more information, see the documentation for the BDT routine you are calling.
- If you specify a BDT name that is not registered with the BDT controller, a trappable runtime error occurs.
- The BDT name is not case sensitive.
- The ErrorCode, ErrorMsg, ErrorPos, and ErrorLen properties are read-only. To determine if a conversion error occurred, you can examine these properties after a call to ConvertFromDisplay and ConvertInPlace.
- The ConvertToDisplay and CreateSampleString methods do not return error information.
- Each BDT conversion routine can define its own error codes and messages. For more information, see the documentation for the BDT routine you are calling.

ConvertFromDisplay Method

This method calls a BDT conversion routine to convert a value from a formatted string to a Visual Basic data type. It returns the raw data value.

Syntax

```
result = object.ConvertFromDisplay(FormattedData, BDTName, _
NatFormatLength)
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object with which BDT routines are registered.
FormattedData	yes	string	Formatted data to convert to a Visual Basic data type.
BDTName	no	string	Name of the BDT to use (may contain modifiers).
NatFormatLength	no	string	Natural format and length.
result		variant	Resulting Visual Basic data type.

See Also

ConvertInPlace Method, page 164, and **ConvertToDisplay Method**, page 165.

ConvertInPlace Method

This method calls a BDT conversion routine to convert a value from a formatted string to a Visual Basic data type and back to a formatted string. It returns the results of the internal call to the ConvertFromDisplay method.

Syntax

```
result = object.ConvertInPlace(FormattedData, BDTName, _
NatFormatLength)
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object with which BDT routines are registered.
FormattedData	yes	string	Formatted data converted to a Visual Basic data type and back.
BDTName	no	string	Name of the BDT to use (may contain modifiers).
NatFormatLength	no	string	Natural format and length.
result		variant	Resulting Visual Basic data type.

Remarks

This method internally calls the ConvertFromDisplay method, passes the results to the ConvertToDisplay method, and then updates the formatted data with the results. Pass the formatted data by reference. If a conversion error occurs, the formatted data is not changed.

See Also

ConvertFromDisplay Method, page 163, and **ConvertToDisplay Method**, page 165.

ConvertToDisplay Method

This method converts a value from a Visual Basic data type to a formatted string value suitable for display to a user. It returns the formatted string.

Syntax

```
result = object.ConvertToDisplay(RawData, BDTName, NatFormatLength)
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object with which BDT routines are registered.
RawData	yes	variant	Raw data to convert to a formatted string.
BDTName	no	string	Name of the BDT to use (may contain modifiers).
NatFormatLength	no	string	Natural format and length.
result		string	Formatted display value.

See Also

ConvertFromDisplay Method, page 163, and **ConvertInPlace Method**, page 164.

CreateSampleString Method

This method creates a representative formatted value for the specified business data type. It returns the formatted string.

Syntax

```
result = object.CreateSampleString(BDTName, NatFormatLength)
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object with which BDT routines are registered.
BDTName	no	string	Name of the BDT to use (may contain modifiers).
NatFormatLength	no	string	Natural format and length.
result		string	Formatted display value.

ErrorCode Property

This property returns the error code from the last conversion error.

Syntax

```
result = object.ErrorCode
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object used to do the conversion.
result		long	Error code.

Remarks

Error codes are defined by the BDT conversion routine. For a complete list of possible error codes, refer to the documentation for the BDT.

If no error occurred, ErrorCode contains 0.

See Also

ErrorLen Property, page 166, **ErrorMsg Property**, page 167, and **ErrorPos Property**, page 168.

ErrorLen Property

This property returns the error length from the last conversion error.

Syntax

```
result = object.ErrorLen
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object used to do the conversion.
result		long	Error length.

Remarks

Optionally, the BDT conversion routine can set this property to inform the calling application of the number of characters in error.

If the BDT conversion routine does not set this property, the property contains 0.

See Also

ErrorCode Property, page 166, **ErrorMsg Property**, page 167, and **ErrorPos Property**, page 168.

ErrorMsg Property

This property returns the error message from the last conversion error.

Syntax

```
result = object.ErrorMsg
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object used to do the conversion.
result		string	Error message.

Remarks

Error messages are defined by the BDT conversion routine. For a complete list of possible error codes and messages, refer to the documentation for the BDT.

The default error message is usually suitable for display to a user, however, you can use the ErrorCode property to look up a custom error message containing context-specific information.

See Also

ErrorCode Property, page 166, **ErrorLen Property**, page 166, and **ErrorPos Property**, page 168.

ErrorPos Property

This property returns the position of the first invalid character in the last conversion error.

Syntax

```
result = object.ErrorPos
```

Part	Required	Data Type	Description
object	yes	BDTController	BDTController object used to do the conversion.
result		long	Position of the first invalid character.

Remarks

Optionally, the BDT conversion routine can set this property to inform the calling application of the position of the first invalid character in the data being converted. The application can use this value in association with the ErrorLen property to highlight the invalid characters for the user.

If the BDT conversion routine does not set this property, the property contains 0.

See Also

ErrorCode Property, page 166, **ErrorLen Property**, page 166, and **ErrorMsg Property**, page 167.

BDTConversion Class

A BDTConversion object is passed into a BDT conversion routine and contains all of the parameters specified by the calling application.

When the application calls one of the conversion routines in the BDT controller, the BDT controller:

- Creates a new BDTConversion object.
- Places the call parameters in the new BDTConversion object.
- Looks up the conversion routine for the specified BDT.
- Passes the BDTConversion object to the conversion routine.

The BDT conversion routine examines the properties of the BDTController object to determine what to do. It places the result of the conversion back in the BDTConversion object before returning to the BDT controller. The BDT controller then returns the result to the application.

The properties and methods used by the BDTConversion class are grouped by function and described in the following sections:

- **Returning Conversion Parameters**, page 170
- **Returning Error Details**, page 179

Returning Conversion Parameters

To return conversion parameters, the BDTConversion class has the following method and properties:

BDTName Property, page 171	**HasModifier Property**, page 175
Conversion Property, page 172	**Length Property**, page 175
Decimals Property, page 173	**Modifier Property**, page 176
Format Property, page 173	**Modifiers Property**, page 177
FormatLength Property, page 174	**RawData Property**, page 177
FormattedData Property, page 174	**SetBDTName Method**, page 178

If the application specifies a Natural format and length in the call:

- FormatLength property contains the entire format and length string
- Format property contains the format portion of the string (a letter)
- Length property contains the length portion of the string (a number)
- Decimals property contains the decimal portion of the string

The individual properties (Format, Length, and Decimals) save the conversion routine from having to parse the format and length string.

BDTName Property

This property returns the name of the business data type (BDT) used.

Syntax

```
result = object.BDTName
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
result		string	Name of the BDT used.

Remarks

The BDT name used in the call is not case sensitive; the caller can use a different case from that used to register the BDT name. However, the BDT controller always fixes the case of the BDT name used so that it matches the name registered. For example:

- The BDT was registered with the name “AccountNo”.
- The application uses the BDT name “ACCOUNTNO” in a call to ConvertToDisplay.
- The BDT controller recognizes this BDT name as “AccountNo” and converts it to mixed case.
- The BDT controller passes “AccountNo” to the conversion routine.

  As a result, the BDT conversion routine does not have to use case-insensitive comparisons when making program flow decisions on this property value.

See Also

ConvertFromDisplay Method, page 163, and **ConvertToDisplay Method**, page 165.

Conversion Property

This property returns the type of conversion to perform.

Syntax

```
result = object.Conversion
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
result		Integer	Type of conversion performed (see the following table).

Setting	Conversion Performed
bdtConvertFromDisplay	Converts the data in FormattedData and places the result in RawData.
bdtConvertToDisplay	Converts the data in RawData and places the result in FormattedData.
bdtCreateSampleString	Creates a sample string and places the result in FormattedData.

Remarks

To perform the conversion appropriate for the value of the Conversion property, the BDT conversion routine generally uses a Select Case statement.

See Also

FormattedData Property, page 174.

Decimals Property

This property returns the decimals portion of the Natural format and length string.

Syntax

```
result = object.Decimals
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
result		Integer	Number of decimal places in the format and length string.

Format Property

This property returns the format portion of the Natural format and length string.

Syntax

```
result = object.Format
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
result		string	Format portion of the Natural format and length string (for example, A for alphanumeric, B for binary).

See Also

FormatLength Property, page 174, and **Length Property**, page 175.

FormatLength Property

This property returns the Natural format and length string passed to the conversion routine in the BDT controller.

Syntax

```
result = object.FormatLength
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
result		string	Format and length string.

See Also

Length Property, page 175, and **Format Property**, page 173.

FormattedData Property

This property gets or sets the formatted data; it is read or write.

Syntax

```
result = object.FormattedData
```

or

```
object.FormattedData = value
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
value		string	Formatted data resulting from the conversion.
result		string	Formatted data to convert to raw data.

Remarks

Depending on the value in the Conversion property, the BDT conversion routine either reads or writes this property.

See Also

Conversion Property, page 172.

HasModifier Property

This property determines whether a specified modifier was used in a call.

Syntax

```
result = object.HasModifier(Modifier)
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
Modifier	yes	string	Name of the modifier to check for.
result		Boolean	Result of the search; this property returns True if the modifier was used, False if the modifier was not.

See Also

Modifier Property, page 176, and **Modifiers Property**, page 177.

Length Property

This property returns the length portion of the Natural format and length string.

Syntax

```
result = object.Length
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
result		Integer	Length portion of the Natural format and length string.

See Also

Format Property, page 173, and **FormatLength Property**, page 174.

Modifier Property

This property returns the name or value of a specified modifier used in a call.

Syntax

```
result = object.Modifier(Modifier)
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
Modifier	yes	variant	See **Remarks**.
result		string	See **Remarks**.

Remarks

You can use this property to:

- Enumerate the modifiers used in the call.

  or

- Obtain the value of a specified modifier used in the call.

  If the Modifier value is numeric, it acts as an ordinal to index the list of modifiers used. In this case, this property returns the name of the modifier in the specified position.

  If the Modifier value is the name of a modifier, this property returns the value of the modifier used in the call.

  For example, the following conversion:

```
strHours = BDT.ConvertToDisplay(dblHours, _
                                "Numeric,ZERO=OFF,ROUND=2,STRICT=ON", _
                                 "N3.2")
```

  Produces the following results:

```
Print .Modifier("ZERO")       ' Prints "OFF"
Print .Modifier("ROUND")      ' Prints "2"
Print .Modifier(1)            ' Prints "ZERO"
Print .Modifier(2)            ' Prints "ROUND"
Print .Modifier(.Modifier(1)) ' Prints "OFF"
```

See Also

HasModifier Property, page 175, and **Modifiers Property**, page 177.

Modifiers Property

This property returns the number of modifiers used in the call to the conversion routine.

Syntax

```
result = object.Modifiers
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
result		Integer	Number of modifiers used.

See Also

HasModifier Property, page 175, and **Modifier Property**, page 176.

RawData Property

This property gets or sets the raw data; it is read or write.

Syntax

```
result = object.RawData
```

or

```
object.RawData = value
```

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
result		variant	Value to convert to a formatted string.
value		variant	Raw data resulting from the conversion.

Remarks

Depending on the value in the Conversion property, the BDT conversion routine either reads or writes this property.

See Also

FormattedData Property, page 174.

SetBDTName Method

This method changes the BDT name or the modifiers or both the BDT name and modifiers in the BDTConversion object.

Syntax

object.SetBDTName *NewName*

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
NewName	yes	string	New BDT name to use.

Remarks

Use this method to call another BDT conversion routine to perform the conversion. To do this, change the BDT name in the BDTConversion object and then call the other conversion routine instead. For example:

```
Public Sub Convert_NumericZeroSuppress(BDTC As BDTConversion)
    ' Change the name of the BDT and add a modifier.
    BDTC.SetBDTName "Numeric,ZERO=OFF"
    ' Call the actual routine to perform the conversion.
    Convert_Numeric BDTC
End Sub
```

Returning Error Details

To return error details, the BDTConversion class has the following properties:

ErrorCode Property, page 179	**ErrorMsg Property**, page 180
ErrorLen Property, page 179	**ErrorPos Property**, page 181

ErrorCode Property

This property sets the error code for the conversion.

Syntax

object.ErrorCode = *value*

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
value	yes	long	Error code.

See Also

ErrorLen Property, page 179, **ErrorMsg Property**, page 180, and **ErrorPos Property**, page 181.

ErrorLen Property

This property sets the error length for the last conversion error.

Syntax

object.ErrorLen = *value*

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
value	yes	long	Error length.

Remarks

Optionally, the BDT conversion routine can set this property to inform the calling application of the number of characters in error.

See Also

ErrorCode Property, page 179, **ErrorMsg Property**, page 180, and **ErrorPos Property**, page 181.

ErrorMsg Property

This property sets the error message for the last conversion error.

Syntax

object.ErrorMsg = *value*

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
value	yes	string	Error message.

Remarks

The default error message should be suitable for display to a user; however, you can use the ErrorCode property to look up a custom error message containing context-specific information.

See Also

ErrorCode Property, page 179, **ErrorLen Property**, page 179, and **ErrorPos Property**, page 181.

ErrorPos Property

This property sets the position of the first invalid character in the last conversion error.

Syntax

object.ErrorPos = *value*

Part	Required	Data Type	Description
object	yes	BDTConversion	BDTConversion object passed to the conversion routine.
value	yes	long	Position of the first invalid character.

Remarks

Optionally, the BDT conversion routine can set this property to inform the calling application of the position of the first invalid character in the data being converted. The application can use this value in association with the ErrorLen property to highlight the invalid characters for the user.

See Also

ErrorCode Property, page 179, **ErrorLen Property**, page 179, and **ErrorMsg Property**, page 180.

COMMAND HANDLER CLASSES

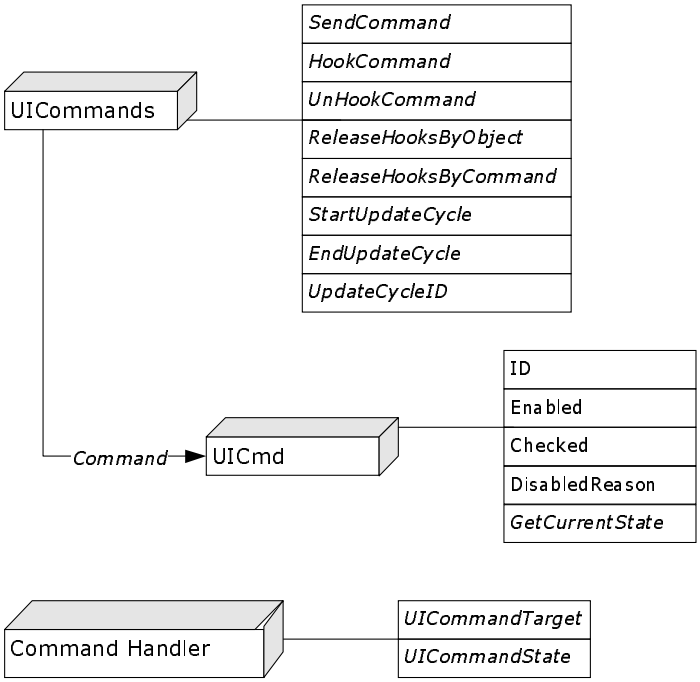
This chapter describes the command handler classes and their methods and properties.

The following topics are covered:

For related information, see:

- **BrowseCommandHandler Class**, page 68

The following diagram shows the structure of the command handler classes:



Structure of the Command Handler Classes

These methods and properties are described in the following sections.

Methods Common to All Command Handlers

The following sections describe the methods that all command handler classes must implement. All command handler objects must have the following methods:

UICommandState Method, page 185	**UICommandTarget Method**, page 187

UICommandState Method

This method is called from the framework and determines whether a command is enabled or disabled, checked or unchecked.

The framework calls this method in the following situations:

- when a user selects a drop-down menu to display another menu
- continuously while an application is idle to determine the state of toolbar buttons

Syntax

```
Public Sub UICommandState(Cmd As UICmd, _
                          ByRef ForwardToNext As Boolean)
```

Part	Required	Data Type	Description
Cmd	yes	UICmd	Object identifying the command the framework is checking.
ForwardToNext	yes	Boolean	Causes the next command handler in the command handler list to be called. The default is False.

Remarks

If the command is disabled, you can also provide a message to be displayed to the user indicating why the command is disabled (as shown in the following example).

Example

```
Public Sub UICommandState(Cmd As UICmd, _
                          ByRef ForwardToNext As Boolean)

    Select Case Cmd.ID
    Case CMD_FILE_CLOSE
        ' The command should only be enabled if the active form is
        ' an MDI child window.
        If IsMDIChild(Screen.ActiveForm) Then
            Cmd.Enabled = True
        Else
            Cmd.Enabled = False
            Cmd.DisabledReason = "This command is not available " & _
                                 "because there are no document " & _
                                 "windows open."
        End If
    Case CMD_TOGGLE_TOOLBAR
        Cmd.Enabled = True
        If tbrMain.Visible Then
            Cmd.Checked = True
        Else
            Cmd.Checked = False
        End If
    Case ...
    ...
End Sub
```

See Also

UICommandTarget Method, page 187.

UICommandTarget Method

This method processes a command; the framework calls this method when a user selects a menu command or toolbar button that is associated with a command linked to the command handler object.

Syntax

```
Public Sub UICommandTarget(Cmd As UICmd, _
                           ByRef ForwardToNext As Boolean)
```

Part	Required	Data Type	Description
Cmd	yes	UICmd	Object identifying the command the user selected.
ForwardToNext	yes	Boolean	Causes the next command handler in the command handler list to be called. The default is False.

Remarks

Typically, you code a Select statement that includes a Case for every command handled by the command handler (as shown in the following example).

Example

```
Public Sub UICommandTarget(Cmd As UICmd, _
                           ByRef ForwardToNext As Boolean)

    Select Case Cmd.ID
    Case CMD_FILE_CLOSE
        ' Close the current window.
        Unload Screen.ActiveForm
    Case CMD_TOGGLE_TOOLBAR
        ' Toggle the visibility of the toolbar.
        tbrMain.Visible = Not tbrMain.Visible
    Case ...
    ...
End Sub
```

See Also

UICommandState Method, page 185.

UICmd Class

This class contains information about a command passed to a command handler object. The UICommandState method for the command handler object sets the Enabled, DisabledReason, and Checked properties for the UICmd object. The UICommandTarget method for the command handler object performs the processing for the command.

To create an instance of the UICmd class, you must call the Command method of the UICommands class.

The UICmd class has the following method and properties:

Checked Property

This property gets or sets the checked or unchecked state of a command; it is read or write.

Syntax

```
object.Checked = value
```

or

```
result = object.Checked
```

Part	Required	Data Type	Description
object	yes	UICmd	Instance of the UICmd class.
value	yes	Boolean	Specified checked or unchecked state.
result		Boolean	Current checked or unchecked state.

Remarks

Typically, you set this property in the UICommandState method of the command handler object. The framework uses the value of this property to set the Checked property of menu items and toolbar buttons.

Example

See the example for the UICommandState method.

See Also

DisabledReason Property, page 189, and **GetCurrentState Method**, page 191.

DisabledReason Property

This property gets or sets the message displayed to a user when a command is disabled; it is read or write.

Syntax

object.DisabledReason = *value*

or

result = *object*.DisabledReason

Part	Required	Data Type	Description
object	yes	UICmd	Instance of the UICmd class.
value	yes	string	Specified message displayed to the user.
result		string	Current message displayed to the user.

Remarks

When you set the Enabled property to False, you can also set this property.

Example

See the example for the UICommandState method.

See Also

Checked Property, page 188, and **GetCurrentState Method**, page 191.

Enabled Property

This property gets or sets the enabled or disabled state of a command; it is read or write.

Syntax

object.Enabled = *value*

or

result = *object*.Enabled

Part	Required	Data Type	Description
object	yes	UICmd	Instance of the UICmd class.
value	yes	Boolean	Specified enabled or disabled state.
result		Boolean	Current enabled or disabled state.

Remarks

Typically, you set this property in the UICommandState method of the command handler object. The framework uses the value of this property to set the Enabled property of menu items and toolbar buttons.

When you set this property to False, you can optionally set the DisabledReason property to a message displayed to the user. The message should indicate why the command is disabled.

Example

See the example for the UICommandState method.

See Also

Checked Property, page 188, and **DisabledReason Property**, page 189.

GetCurrentState Method

This method directs the framework to call the UICommandState method of the command handler object associated with the command.

Syntax

object.GetCurrentState

Part	Required	Data Type	Description
object	yes	UICmd	Instance of the UICmd class.

Remarks

After returning from this method, you can check the Enabled, DisabledReason, and Checked properties to determine the state of the command.

Example

```
Private Sub SetMenuState(mnu As Menu, CmdID As Variant)
    With gUICmds.Command(CmdID)
        .GetCurrentState
        mnu.Enabled = .Enabled
        mnu.Checked = .Checked
    End With
End Sub
```

See Also

Checked Property, page 188, and **DisabledReason Property**, page 189.

ID Property

This property gets the command ID associated with a command; it is read-only.

Syntax

```
result = object.ID
```

Part	Required	Data Type	Description
object	yes	UICmd	Instance of the UICmd class.
result		variant	Command ID.

Example

See the example for the UICommandState method.

UICommands Class

This class is the main driver for the command handling mechanism in the Spectrum client framework.

The UICommands class has the following methods and properties:

Command Method, page 193	**SendCommand Method**, page 198
EndUpdateCycle Method, page 194	**StartUpdateCycle Method**, page 199
HookCommand Method, page 195	**UnHookCommand Method**, page 200
ReleaseHooksByCommand Method, page 196	**UpdateCycleID Property**, page 201
ReleaseHooksByObject Method, page 197	

Command Method

This method creates an instance of the UICmd class; use the UICmd object to call the GetCurrentState method.

Syntax

```
Set result = object.Command(CmdID)
```

Part	Required	Data Type	Description
object	yes	UICommands	Instance of the UICommands class.
CmdID	yes	variant	Command ID of a command.
result		UICmd	Instance of the UICmd class.

Remarks

Because the UICmd class cannot be created externally, you must use this method to create instances of the class.

Example

```
Dim cmd As UICmd

Set cmd = gUICmds.Command(CMD_FILE_CLOSE)
cmd.GetCurrentState
mnuFileClose.Enabled = cmd.Enabled
mnuFileClose.Checked = cmd.Checked
```

EndUpdateCycle Method

This method ends an update cycle.

Syntax

object.EndUpdateCycle

Part	Required	Data Type	Description
object	yes	UICommand	Instance of the UICommands class.

Remarks

Use the StartUpdateCycle method and this method to bracket multiple calls to the GetCurrentState method of the UICmd class. Within an update cycle, command handler objects can assume that the state of an application does not change.

To obtain a unique identifier for the current update cycle, use the UpdateCycleID property within the UICommandState method of the command handler object.

Example

```
Private Sub mnuFile_Click()
    gUICmds.StartUpdateCycle
    SetMenuState mnuFileOpen, CMD_FILE_OPEN
    SetMenuState mnuFileClose, CMD_FILE_CLOSE
    gUICmds.EndUpdateCycle
End Sub
```

See Also

StartUpdateCycle Method, page 199, and **UpdateCycleID Property**, page 201.

HookCommand Method

This method associates command IDs with command handler objects.

Syntax

object.HookCommand *HookObject, CmdID...*

Part	Required	Data Type	Description
object	yes	UICommand	Instance of the UICommands class.
HookObject	yes	object	Reference to a command handler object.
CmdID	yes	variant	One or more command IDs to associate with the command handler object.

Remarks

This method adds the command handler object to the list of command handler objects for each command ID specified.

Example

```
gUICmds.HookCommand Me, CMD_FILE_OPEN, CMD_FILE_CLOSE
```

See Also

- **ReleaseHooksByCommand Method**, page 196
- **ReleaseHooksByObject Method**, page 197
- **SendCommand Method**, page 198
- **UnHookCommand Method**, page 200

ReleaseHooksByCommand Method

This method clears the command handler list for a command ID.

Syntax

object.ReleaseHooksByCommand *CmdID*

Part	Required	Data Type	Description
object	yes	UICommand	Instance of the UICommands class.
CmdID	yes	variant	Command ID.

Remarks

Use this method to ensure that no other command handler objects are associated with the command ID.

Example

```
gUICmds.ReleaseHooksByCommand CMD_FILE_CLOSE
```

See Also

HookCommand Method, page 195, **ReleaseHooksByObject Method**, page 197, and **UnHookCommand Method**, page 200.

ReleaseHooksByObject Method

This method removes an object from all command handler lists.

Syntax

object.ReleaseHooksByObject *HookObject*

Part	Required	Data Type	Description
object	yes	UICommand	Instance of the UICommands class.
HookObject	yes	object	Command handler object to remove from all command handler lists.

Remarks

Use this method when an object that handles commands (such as a form) is deleted.

Example

```
gUICmds.ReleaseHooksByObject Me
```

See Also

HookCommand Method, page 195, **ReleaseHooksByCommand Method**, page 196, and **UnHookCommand Method**, page 200.

SendCommand Method

This method causes the UICommandTarget method of a command handler object to be called to process a command ID.

Syntax

object.SendCommand *CmdID*

Part	Required	Data Type	Description
object	yes	UICommand	Instance of the UICommands class.
CmdID	yes	variant	Command ID.

Remarks

This method calls the first command handler object on the command handler list. To process the command, use the UICommandTarget method of the specified command handler object.

The command handler can also set the ForwardToNext parameter to True to cause the next command handler object on the command handler list to be called.

Example

```
gUICmds.SendCommand CMD_FILE_NEW
```

See Also

- **HookCommand Method**, page 195
- **ReleaseHooksByCommand Method**, page 196
- **ReleaseHooksByObject Method**, page 197
- **UnHookCommand Method**, page 200

StartUpdateCycle Method

This method starts an update cycle.

Syntax

object.StartUpdateCycle

Part	Required	Data Type	Description
object	yes	UICommand	Instance of the UICommands class.

Remarks

Use this method and the EndUpdateCycle method to bracket multiple calls to the GetCurrentState method of the UICmd class. Within an update cycle, command handler objects can assume that the state of an application does not change.

To obtain a unique identifier for the current update cycle, use the UpdateCycleID property within the UICommandState method of the command handler object.

Example

```
Private Sub mnuFile_Click()
    gUICmds.StartUpdateCycle
    SetMenuState mnuFileOpen, CMD_FILE_OPEN
    SetMenuState mnuFileClose, CMD_FILE_CLOSE
    gUICmds.EndUpdateCycle
End Sub
```

See Also

EndUpdateCycle Method, page 194, and **UpdateCycleID Property**, page 201.

UnHookCommand Method

This method removes a command handler object from the command handler list for one or more command IDs.

Syntax

```
object.UnHookCommand HookObject, CmdID...
```

Part	Required	Data Type	Description
object	yes	UICommand	Instance of the UICommands class.
HookObject	yes	object	Reference to a command handler object.
CmdID	yes	variant	One or more command IDs.

Remarks

Use this method when you know exactly which command IDs and command handler objects you want to remove.

Example

```
gUICmds.UnHookCommand Me, CMD_FILE_OPEN, CMD_FILE_CLOSE
```

See Also

- **HookCommand Method**, page 195
- **ReleaseHooksByCommand Method**, page 196
- **ReleaseHooksByObject Method**, page 197
- **SendCommand Method**, page 198

UpdateCycleID Property

This property gets a unique identifier for each update cycle.

Syntax

```
result = object.UpdateCycleID
```

Part	Required	Data Type	Description
object	yes	UICommand	Instance of the UICommands class.
result		long	Current update cycle identifier.

Remarks

Command handler objects can use this property to determine when a new update cycle starts. Once a new cycle begins, they can gather information about the state of the application at the start of the update cycle; they can use this information throughout the update cycle to assign the Enabled and Checked properties of the UICmd objects.

Example

```
Public Sub UICommandState(Cmd As UICmd, ForwardToNext As Boolean)

    Dim ctl As Control
    Static llastupdateid As Long
    Static bselected As Boolean
    Static bcanpaste As Boolean

    With Cmd
        Select Case .ID
        Case CMD_EDIT_CUT, CMD_EDIT_COPY, CMD_EDIT_PASTE
            If llastupdateid <> gUICmds.UpdateCycleID Then
                llastupdateid = gUICmds.UpdateCycleID

                bselected = False
                bcanpaste = False
                Set ctl = Screen.ActiveControl
                If Not (ctl Is Nothing) Then
                    If TypeOf ctl Is TextBox Then
                        bselected = (ctl.SelLength > 0)
                        bcanpaste = Clipboard.GetFormat(vbCFText)
                    End If
                End If
            End If

            Select Case .ID
            Case CMD_EDIT_CUT:   .Enabled = bselected
            Case CMD_EDIT_COPY:  .Enabled = bselected
            Case CMD_EDIT_PASTE: .Enabled = bcanpaste
            End Select
        ...
        End Select
    End With

End Sub
```

See Also

EndUpdateCycle Method, page 194, and **StartUpdateCycle Method**, page 199.

MAINTENANCE CLASSES

This chapter describes the maintenance classes, which cooperate with a maintenance dialog and other utilities to provide maintenance functionality. It includes information about the generated Visual Basic maintenance object (created using the VB-Maint-Object model), the ComboClass class, and the TrueGridClass class.

The following topics are covered:

Note: For information on using a grid in “Unbound” mode, see the online help.

Generated Visual Basic Maintenance Object

This section describes the generated Visual Basic maintenance object, which provides a common interface to its related maintenance object on the server.

The generated Visual Basic maintenance object has the following methods and properties:

BrowseByForeignKey Method, page 205	**GetAllForeignKeyValues Method**, page 212
BrowseByObjectKey Method, page 206	**GetField Method**, page 213
ContainsDerivedData Property, page 207	**InvokeMethod Method**, page 214
Dispatcher Property, page 207	**KeyChanged Method**, page 214
Exists Property, page 208	**MoveByNameKey Method**, page 215
Field Property, page 209	**Msg Property**, page 215
FieldDef Property, page 209	**Name Property**, page 216
ForeignKeyBrowserExists Property, page 210	**ObjectDataArea Property**, page 216
ForeignKeyCount Property, page 211	**ObjectKeyBrowserExists Property**, page 217
ForeignKeyLookup Method, page 211	**SetField Method**, page 218

BrowseByForeignKey Method

This method displays a modal browse dialog that can be used to browse a foreign table linked to a specified relationship name.

Syntax

object.BrowseByForeignKey *RelationshipName*, *Dim1*, *Dim2*

Part	Required	Data Type	Description
RelationshipName	yes	object	Name of a foreign key relationship defined in Predict.
Dim1, 2	no	Integer	Indices that uniquely identify occurrences of vector data.

Remarks

If a new foreign key value is selected on the browse dialog, the object data parameter data area is updated with the selected value. The selected row is returned in a Browse-DataCache object.

Example

```
Relationship = ProductGrid.ForeignRelationship(ColIndex)
Row = ProductGrid.CurrentRow
Set BrCache = InternalObject.BrowseByForeignKey(Relationship, _
                                                Row)
```

See Also

BrowseByObjectKey Method, page 206.

BrowseByObjectKey Method

This method displays a modal browse dialog that can be used to select a new object key.

Syntax

object.BrowseByObjectKey

Remarks

If a new object key is selected, this method returns True and the KeyData property contains the value of the selected object key. You can then use the Get request to retrieve the object data associated with the key value. If a new object key is not selected, this method returns False.

As illustrated in the example, it is a good idea to release the hooked commands before calling this method and then re-hook them after control returns from the modal browse dialog. This ensures that pop-up menus that do not pertain to the dialog are not accessible from the dialog when a user clicks the right mouse button.

Example

```
Case ACTION_BROWSE
    ' Unhook all MDI frame commands.
    gUICmds.ReleaseHooksByObject Me

    ' Invoke a modal browse dialog.
    bSuccess = InternalObject.BrowseByObjectKey

    ' Rehook commands.
    HookCommands
```

See Also

BrowseByForeignKey Method, page 205.

ContainsDerivedData Property

This property gets a Boolean value indicating whether a specified object contains derived data; it is read-only.

Syntax

```
result = object.ContainsDerivedData
```

Remarks

The Visual Basic maintenance dialog associated with a Visual Basic maintenance object uses this property to determine whether it needs to refresh the contents of its GUI controls after a successful Update or Add request.

Example

```
If InternalObject.ContainsDerivedData Then
    CopyObjectToForm
End If
```

Data Type

Boolean

Dispatcher Property

This property gets or sets a reference to a Dispatcher object that services requests for a specified business object; it is read or write.

Syntax

```
Set object.Dispatcher = value
```

or

```
Set result = object.Dispatcher
```

Remarks

Dispatcher objects are created in the Visual Basic maintenance dialog using the CreateDispatcher method — rather than in the Visual Basic maintenance object. This allows the dialog to use a single Dispatcher object to serve two or more Visual Basic maintenance objects. To set up transactions that span Visual Basic maintenance objects, you provide custom code in the Visual Basic maintenance dialog.

Example

```
Set InternalObject.Dispatcher = CreateDispatcher()
```

Data Type

Dispatcher object

Exists Property

This property gets a Boolean value indicating whether a specified business object exists; it is read-only.

Syntax

value = object.Exists

Remarks

This property is derived from the m_CDAOBJ parameter data area (PDA), which is passed back and forth between the client and server. Each time a server method request is made, the associated object subprogram sets this Boolean variable. For example, if a user requests a record that does not exist in the database, this property returns False. If a user requests a record that already exists, this property returns True.

Note: This property is not currently used by Visual Basic maintenance dialogs.

Data Type

Boolean

Field Property

This property gets or sets the value of a specified field in a NaturalDataArea object instantiated by the Visual Basic maintenance object; it is read or write.

Syntax

object.Field(*FieldSpec*)

Part	Required	Data Type	Description
FieldSpec	yes	string	Name of a field in a NaturalDataArea object.

Remarks

This property shares the same name and purpose as the Field property of a NaturalDataArea object; this makes it possible to provide polymorphic behavior. The caller can reference either a Visual Basic maintenance object or a NaturalDataArea object — and manipulate the Field property of that object. The difference between the two property procedures is that this property performs several layers of validation on the value before assigning it to the NaturalDataArea object.

Example

```
cbo_Empl_MarStat.ListIndex = _
           MarStatList.GetIndex(InternalObject.Field("MAR-STAT"))
```

Data Type

Variant

FieldDef Property

This property gets a Natural field definition for a specified field specification; it is read-only.

Syntax

object.FieldDef(*FieldSpec*)

Part	Required	Data Type	Description
FieldSpec	yes	string	Field specification.

Remarks

Currently, only the ValidAssignment function in BDTSupport.bas uses this property.

Data Type

NaturalFieldDef object

See Also

ValidAssignment Function, page 401, and **Utility Subroutines on the Client**, page 369.

ForeignKeyBrowserExists Property

This property gets a Boolean value indicating whether a foreign key browser exists for a specified relationship; it is read-only.

Syntax

object.ForeignKeyBrowserExists (*Rel*)

Part	Required	Data Type	Description
Rel	yes	string	Name of the relationship.

Remarks

Use this property to dynamically enable or disable foreign key support on a Visual Basic maintenance dialog (based on the existence of supporting browser code).

Example

```
FKeyRelName = "NCST-POLICY-IS-FOR-CUSTOMER"
If Not InternalObject.ForeignKeyBrowserExists(FKeyRelName) Then
    cmd_InsM_NcstPolicyIsForCustomer.Enabled = False
    cmd_InsM_NcstPolicyIsForCustomer.Visible = False
End If
```

Data Type

Boolean

ForeignKeyCount Property

This property gets an Integer indicating the number of foreign keys linked to a Visual Basic maintenance object; it is read-only.

Syntax

```
result = object.ForeignKeyCount
```

Remarks

Currently, this property is not used outside of the Visual Basic maintenance object.

Data Type

Integer

ForeignKeyLookup Method

This method looks up a specified foreign key value on a foreign table and gets a reference to a BrowseDataCache object containing the column values for that row.

Syntax

```
result = object.ForeignKeyLookup(RelationshipName, Dim1, Dim2)
```

Part	Required	Data Type	Description
RelationshipName	yes	string	Name of the foreign key relationship.
Dim1, 2	no	variant	Occurrence(s) used when assigning the value of the foreign key.

Data Type

BrowseDataCache object

Example

```
Set BrCache = _
       InternalObject.ForeignKeyLookup("NCST-WAREHOUSE-ORDER-HEADER")
With BrCache
    lbl_OrdM_NcstWarehouseOrderHeader.Caption = _
    BDT.ConvertToDisplay(.Rows.Item(1).Value("WAREHOUSE-DESC"), _
                         .Columns.Item("WAREHOUSE-DESC").BDT, _
                         .Columns.Item("WAREHOUSE-DESC").Format)
End With
```

GetAllForeignKeyValues Method

This method returns a reference to a BrowseDataCache object containing all the rows in a foreign file.

Syntax

result = object.GetAllForeignKeyValues(*RelationshipName*)

Part	Required	Data Type	Description
RelationshipName	yes	string	Name of the relationship.

Remarks

Use this method when the foreign file is relatively small and stable. This allows you to place foreign key values in a combobox control on the Visual Basic maintenance dialog.

GetField Method

This method returns a variant containing the value of a specified field in a Natural-DataArea object.

Syntax

```
result = object.GetField(FieldSpec, Index1, Index2, Index3)
```

Part	Required	Data Type	Description
FieldSpec	yes	string	Field specification.
Index1, 2, 3	no	variant	Indices that uniquely identify occurrences of vector data.

Remarks

This method shares the same name and purpose as the GetField method of a Natural-DataArea object; this makes it possible to provide polymorphic behavior. The caller can reference either a Visual Basic maintenance object or a NaturalDataArea object — and call the GetField method of that object.

Example

```
IncomeGrid.GridData(i, 1) = _
      BDT.ConvertToDisplay(InternalObject.GetField("CURR-CODE", i), _
      NatFormatLength:="A3")
```

See Also

SetField Method, page 218, and **Field Property**, page 209.

InvokeMethod Method

This method invokes a method on behalf of a Visual Basic maintenance dialog and returns a Boolean value indicating the success or failure of the method.

Syntax

```
result = object.InvokeMethod(MethodName, Flags)
```

Part	Required	Data Type	Description
MethodName	yes	string	Name of the method.
Flags	no	variant	Method-specific flags.

Remarks

This method does not implement method requests made by the Visual Basic maintenance object. Rather, it determines whether the method is implemented locally or remotely and then calls the private methods of the Visual Basic maintenance object (which implement local and remote method requests).

Example

```
Case ACTION_GET
    bSuccess = InternalObject.InvokeMethod("GET")
    If bSuccess Then CopyObjectToForm
```

KeyChanged Method

This method returns a Boolean value indicating whether the object key value has been modified.

Syntax

```
result = object.KeyChanged()
```

Remarks

The Visual Basic maintenance dialog uses this property to determine whether Add, Update, and Delete requests are available to the user (based on whether the key value has changed).

Example

```
' check whether the Add method should be allowed
If Not InternalObject.KeyChanged Then
    ErrorMsg = csterrInvalidAddMsg
    CheckUIStatus = False
End If
```

MoveByNameKey Method

This method simulates a Natural MOVE-BY-NAME statement, moving key information from the object data structure to object key structure (or vice versa) of an object PDA (implemented through a NaturalDataArea object).

Syntax

object.MoveByNameKey(*Direction*)

Part	Required	Data Type	Description
Direction	no	Integer	Indicates whether key information is moved from the object data structure to the object key structure or vice versa. Use the MOVE_DATA_TO_KEY or MOVE_KEY_TO_DATA constant.

Remarks

Currently, this method is only used internally by the Visual Basic maintenance object.

Msg Property

This property gets a references to a NaturalDataArea object containing message information; it is read-only.

Syntax

Set result = *object*.Msg

Remarks

Internally, this NaturalDataArea object refers to the CDPDA-M Natural data area. This data area is populated on the server.

Example

```
'error isn't associated with a specific GUI control
If ErrControl Is Nothing Then
    MsgBox cstFormatMessage(InternalObject.Msg), vbInformation
End If
```

Data Type

NaturalDataArea object

See Also

cstFormatMessage Function, page 377, and **Utility Subroutines on the Client**, page 369.

Name Property

This property gets the name of an object parameter data area (PDA).

Syntax

`result = object.Name`

Remarks

This property is not currently used by the Visual Basic maintenance dialog.

Data Type

String

ObjectDataArea Property

This property gets a reference to a NaturalDataArea object representing an object parameter data area (PDA).

Syntax

`Set result = object.ObjectDataArea`

Remarks

This property is not currently used by Visual Basic maintenance dialogs.

Data Type

NaturalDataArea object

ObjectKeyBrowserExists Property

This property gets a Boolean value indicating whether an object key browser exists.

Syntax

```
result = object.ObjectKeyBrowserExists
```

Remarks

This property is based on the BrowserExists subroutine in the object factory module. Use this property to determine whether a browse component for a business object has been generated.

Example

```
' If the ObjectKeyBrowser used by this dialog exists,
' then hook the browse command.
If InternalObject.ObjectKeyBrowserExists Then
    gUICmds.HookCommand Me, CMD_ACTIONS_BROWSE
End If
```

Data Type

Boolean

See Also

Object Factory, page 263.

SetField Method

This method updates a field in a NaturalDataArea object with a specified value.

Syntax

object.SetField *Value*, *FieldSpec*, *Index1*, *Index2*, *Index3*

Part	Required	Data Type	Description
Value	yes	variant	Value assigned to the specified field.
FieldSpec	yes	string	Field specification.
Index1, 2, 3	no	variant	Indices to uniquely identify occurrences of vector data.

Remarks

This method shares the same name and purpose as the SetField method of a Natural-DataArea object; this makes it possible to provide polymorphic behavior. The caller can reference either a Visual Basic maintenance object or a NaturalDataArea object — and call the SetField method of that object. The difference between the two methods is that this method performs several layers of validation on the value before assigning it to the NaturalDataArea object.

Example

```
InternalObject.SetField Value, "INCOME", IncomeNdx
```

See Also

GetField Method, page 213, and **Field Property**, page 209.

ComboClass Class

This class provides a common interface for defining finite sets of selections that can be used by a combobox control or a True DBGrid combobox column. It is useful for representing foreign key selection lists.

Each selection set is comprised of a database value (DBValue) and a display value. The class builds an association between the index value of an item in a combobox list and a DBValue. You can set the Sorted property for the combobox to True and the class still returns the correct value.

The ComboClass class has the following methods and properties:

AddItem Method

This method adds a database-value pair and a display-value pair to a selection list maintained by the ComboClass class.

Syntax

object.AddItem (*DBValue, DisplayValue*)

Part	Required	Data Type	Description
DBValue	yes	variant	Database value.
DisplayValue	no	variant	Display value. The default is DBValue.

Remarks

Before using this method, call the Load method to initialize an instance of the ComboClass class.

Example

```
MProvinceList.Load cbo_CUST_Mprovince
MProvinceList.AddItem "British Columbia"
MProvinceList.AddItem "Alberta"
MProvinceList.AddItem "Saskatchewan"
```

See Also

Load Method, page 221.

DBValue Property

This property returns a database value associated with a display value in a selection list. The Index parameter identifies the database value.

Syntax

object.DBValue *Index*

Part	Required	Data Type	Description
Index	yes	Integer	Index value identifying an item in the selection list.

Remarks

This property is read-only and should not be accessed until the Load method and AddItems methods for the ComboClass class are invoked.

Example

```
result = MProvList.DBValue(cbo_MProv.ItemData(cbo_MProv.ListIndex))
```

Data Type

Variant

GetIndex Method

This method returns an index value pointing to an item in a ComboClass object's selection list for a specified database value.

Syntax

object.GetIndex (*DBValue*)

Part	Required	Data Type	Description
DBValue	yes	variant	Database value.

Remarks

This method is not supported for BrowseDataColumn objects.

Example

```
cbo_CUST_MProvince.ListIndex = _
                    MProvinceList.GetIndex(.Field("M-PROVINCE"))
```

Data Type

Integer

Load Method

This method initializes an instance of the ComboClass class, recording what type of object (combobox or grid column) is defined by the class.

Syntax

object.Load (*ComboControl*)

Part	Required	Data Type	Description
ComboControl	yes	object	Reference to a combo box control or a grid column object (an object exposed by the True DBGrid OCX).

Remarks

Call this method before making calls to any other method used by the ComboClass class.

Example

```
MProvinceList.Load cbo_CUST_MProvince
```

TrueGridClass Class

This class encapsulates the functionality of the True DBGrid; it provides a simpler interface specifically designed for a Spectrum application.

The TrueGridClass class has the following methods and properties:

AfterColEdit Method

This method transfers cell values from the internal buffer for the True DBGrid to the GridData array property.

Syntax

object.AfterColEdit *RowIndex, ColIndex*

Part	Required	Data Type	Description
RowIndex	yes	Integer	Updated grid row.
ColIndex	yes	Integer	Updated grid column.

Remarks

This method is called from within the AfterColEdit event code for a grid on a form. It adjusts the column index value if the grid is initialized without a counter column. The counter column accounts for the discrepancy between the zero-based column array for the grid and the one-based storage array for the TrueGridClass class.

Example

```
ProductsGrid.AfterColEdit ProductsRow, ColIndex
```

BackColor Property

This property returns the RGB background color value for the TrueGridClass class.

Syntax

object.BackColor = *colorvalue*

Remarks

This property provides the ObjectErrors class with polymorphic behavior. The ObjectErrors class stores a reference to either a GUI control or TrueGridClass class. The ObjectErrors class interrogates this object reference for the BackColor property. To fulfill this request, the BackColor property procedure returns the RGB value stored in the ModifiableBackColor constant.

The ObjectErrors class attempts to assign a value to the BackColor property when an ObjectError object is removed. Because the TrueGridClass class does not implement "visual" properties, the value passed to the Property Let procedure is ignored.

Example

```
EmployeeGrid.BackColor = ObjError.RegBackColor ‘ no effect
Temp BackColor = EmployeeGrid.BackColor ‘ returns contents of _
                                        ‘ ModifiableBackColor constant
```

Data Type

Long

See Also

FetchCellStyle Method, page 229, **ForeColor Property**, page 230, and **Object Error Classes**, page 247.

ColumnAdd Method

This method adds a column to a grid at runtime.

Syntax

```
object.ColumnAdd Caption, BDT, FormatLength, Occurrences, _
            Modifiable, Presentation, RowCounter, DropDown _
            ForeignRelationshipName
```

Part	Required	Data Type	Description
Caption	yes	string	Column heading.
BDT	yes	string	Business data type associated with this column.
FormatLength	yes	string	Natural format and length of the data field displayed in the column.
Occurrences	no	Integer	If the data field is an array, the number of occurrences of the array. The default is 1.
Modifiable	no	Boolean	Indicates whether the column is user-modifiable (True) or read-only (False). The default is True.
Presentation	no	Integer	Presentation value. Use the dbgNormal, dbgComboBox, dbgRadioButton, and dbgSortedComboBox constants. The default is dbgNormal.

Part	Required	Data Type	Description (continued)
RowCounter	no	Boolean	Indicates whether the column represents the counter column for the grid. The default is False.
DropDown	no	Boolean	Indicates whether the column acts as a placeholder for a drop-down grid. The default is False.
ForeignRelationship Name	no	string	Name of the foreign key relationship that links the field associated with this column to a foreign file.

Remarks

- If the Caption argument contains slashes (/), the slashes represent line breaks in the column heading text.
- The BDT and FormatLength arguments help determine the width of the column. A sample string is created and applied to the ConvertToDisplay method for the business data type. The calculated column width is wide enough to hold the sample string. If the BDT is set to BDT_BOOLEAN, the column displays a check box.
- If the DropDown argument is True, this method sets up the column to act as a placeholder for a drop-down grid. In addition, the column cells display a button and are non-modifiable.
- If the ForeignRelationshipName argument is not blank, this method configures the grid column to display a button. Selecting this button invokes a modal browse dialog that displays records from a foreign file for selection.

Note: You must call the Load method before calling the ColumnAdd method.

Example

```
Private Sub LoadNcstOrderHasLinesGrid()
    With NcstOrderHasLinesGrid
        Set .Grid = grd_ORD_NcstOrderHasLines
        Set .GridForm = Me
        .Load MAX_NCSTORDERHASLINES_ROWS
        .ColumnAdd "Line/Number", BDT_NUMERIC, "N2"
        .ColumnAdd "Product/Group", BDT_ALPHA, "A2"
        .ColumnAdd "Quantity", BDT_NUMERIC, "P9"
        .ColumnAdd "Unit/Cost", BDT_CURRENCY, "P7.2"
        .ColumnAdd "Total/Cost", BDT_CURRENCY, "P9.2", _
                   Modifiable:=False
        .SetWidth True, True
    End With
End Sub
```

See Also

Load Method, page 221, **Grid Property**, page 231, and **GridForm Property**, page 233.

ColumnFormat Property

This property returns a value indicating the type of format applied to a grid column; it is read-only.

Syntax

`result = object.ColumnFormat(ColIndex)`

Part	Required	Data Type	Description
ColIndex	yes	Integer	Grid column for which you want to determine the format type.

Remarks

This property is read-only. The returned value corresponds to one of the constants prefixed with GCF (found in CSTObjectConsts.bas). To compare the returned value, use these constants.

Example

```
If ProductGrid.ColumnFormat(ColIndex)= GCF_BOOLEAN Then
     MsgBox “Column is boolean”
End If
```

Data Type

Integer

CurrentRow Method

This method returns the currently selected grid row.

Syntax

```
result = object.CurrentRow
```

Remarks

This method returns a one-based value. If the first row in the grid is selected, this method returns 1. Column headings are not considered as a grid row. If there was an error in setting the CurrentRow method, the method returns –1.

To determine the validity of the returned row value, call the NotInBounds method after this method.

Example

```
CurrRow = NcstLineHasDistributionGrid.CurrentRow
If NcstLineHasDistributionGrid.NotInBounds(CurrRow) Then
    ValidateGridAction = False
    Exit Function
End If
```

Data Type

Integer

See Also

NotInBounds Method, page 239.

DeleteGridRow Method

This method deletes the data for the currently selected row on a grid.

Syntax

object.DeleteGridRow

Remarks

When this method is invoked, the number of rows available to the grid is not reduced by one. Instead, data for the selected row is deleted and data for all subsequent rows is moved up by one row. This method works with the GridData property.

Example

```
NcstOrderHasLinesGrid.DeleteGridRow
```

See Also

InsertGridRow Method, page 236, and **GridData Property**, page 232.

DropDownGrid Method

This method drops down a child grid from the grid associated with this instance of the TrueGridClass class.

Syntax

object.DropDownGrid *dd_grid, MuOcc*

Part	Required	Data Type	Description
dd_grid	yes	TrueGridClass	Reference to the TrueGridClass class associated with the drop-down (child) grid.
MuOcc	no	Integer	Currently selected row in the drop-down (child) grid. The default is 1.

Remarks

The method handles all the positioning and sizing details for the drop-down (child) grid. If an MuOcc value is specified, this method sets the selected record in the drop-down grid to the one indicated by the MuOcc value. The drop-down grid represents a single repeating field within another grid.

Example

```
ProductsGrid.DropDownGrid DescriptionGrid
```

EraseData Method

This method erases (removes) all data from a grid.

Syntax

object.EraseData

Remarks

This method resets all occurrences of the GridData property to their default (empty) value. For numeric fields, the default is 0; for all other fields, the default is blank.

Example

```
NcstOrderHasLinesGrid.EraseData
```

FetchCellStyle Method

This method sets the foreground (ForeColor property) and background (BackColor property) color for individual grid cells based on whether the cells are in error.

Syntax

object.FetchCellStyle *CellStyle*, *Col*, *Index1*, *Index2*, *Index3*

Part	Required	Data Type	Description
CellStyle	yes	object	Style object containing the ForeColor and BackColor properties to be set.
Col	yes	Integer	Grid column for which you are setting color properties.
Index1	yes	Integer	Unique occurrence of an array field.
Index2	no	Integer	Unique occurrence of an array field.
Index3	no	Integer	Unique occurrence of an array field.

Remarks

This method is typically called from the FetchCellStyle event code for a grid on a form. The method recognizes the following “color” states:

- in error
- not in error; modifiable
- not in error; non-modifiable

These color states are determined based on the attached ObjectError objects and the column attributes that were established when the grid was initialized.

Example

```
Private Sub grd_ORD_Products_FetchCellStyle(ByVal Condition _
            As Integer, ByVal Split As Integer, _
            Bookmark As Variant, ByVal Col As Integer, _
            ByVal CellStyle As Object)
    Dim ProductsRow As Integer
    ProductsRow = Bookmark + 1
    ProductsGrid.FetchCellStyle CellStyle, Col, ProductsRow
End Sub
```

See Also

BackColor Property, page 223, and **ForeColor Property**, page 230.

ForeColor Property

This property returns the RGB foreground color value for the TrueGridClass class.

Syntax

object.ForeColor = *colorvalue*

Remarks

This property provides the ObjectErrors class with polymorphic behavior. The ObjectErrors class stores a reference to either a GUI control or TrueGridClass class instance as an object. The ObjectErrors class interrogates this object reference for the ForeColor property. To fulfill this request, the ForeColor property procedure returns the RGB value stored in the ModifiableForeColor constant.

The ObjectErrors class attempts to assign a value to the ForeColor property when an ObjectError object is removed. Because the TrueGridClass class does not implement “visual” properties, the value passed to the Property Let procedure is ignored.

Example

```
EmployeeGrid.ForeColor = ObjError.RegForeColor ‘ no effect
Temp ForeColor = EmployeeGrid.ForeColor ‘ returns contents of _
                                        ‘ ModifiableForeColor constant
```

Data Type

Long

See Also

FetchCellStyle Method, page 229, **BackColor Property**, page 223, and **Object Error Classes**, page 247.

ForeignRelationship Property

This property accepts an Integer identifying a grid column and returns the name of the foreign (inter-object) relationship defined for that column; it is read-only.

Syntax

`result = object.ForeignRelationship (Col)`

Part	Required	Data Type	Description
Col	yes	Integer	Grid column for which to return the foreign relationship name.

Data Type

String

Grid Property

This property contains a reference to the underlying True DBGrid control encapsulated by the TrueGridClass class.

Remarks

This property is used by the ErrorTip class to determine the absolute coordinates of a specified grid cell. Because the reference to a grid control is directly accessible on a Visual Basic maintenance dialog, this property is not generally used from a maintenance dialog.

Example

```
…
p_ErrLeft = p_ErrLeft + ObjError.ErrControl.Grid.Columns(l_Col).Left
p_ErrTop = p_ErrTop + _
           ObjError.ErrControl.Grid.RowTop(l_Row - _
           ObjError.ErrControl.Grid.FirstRow)
m_ErrWidth = ObjError.ErrControl.Grid.Columns(l_Col).Width
m_ErrHeight = ObjError.ErrControl.Grid.RowHeight
…
```

Data Type

True DBGrid

See Also

- **ColumnAdd Method**, page 224
- **Load Method**, page 221
- **SetWidth Method**, page 242
- **GridForm Property**, page 233
- **Object Error Classes**, page 247

GridData Property

This property stores array data that is used to populate a True DBGrid. It is used by the ReadGridData, UnboundReadData, UnboundWriteData, and WriteGridData methods.

Syntax

object.GridData(*Row*, *Col*) = Value

or

result = *object*.GridData(*Row*, *Col*)

Part	Required	Data Type	Description
Row	yes	Integer	Grid row to populate with array data.
Col	yes	Integer	Grid column to populate with array data.

Example

```
grd_Empl_Income.Columns(4).Value = "(" & BonusRow & ") " & _
                   BonusGrid.GridData(BonusRow, 1)
```

Data Type

String

See Also

- **ReadGridData Method**, page 240
- **UnboundReadData Method**, page 243
- **UnboundWriteData Method**, page 244
- **WriteGridData Method**, page 245

GridForm Property

This property contains a reference to the form that declared an instance of the True DBGrid control and its associated TrueGridClass class.

Syntax

```
object.GridForm
```

Remarks

This property is set by the form that declared an instance of the TrueGridClass class. The form sets this property when it initializes the grid. The TrueGridClass class uses this reference internally.

Example

```
Set NcstOrderHasLinesGrid.GridForm = Me
```

Data Type

Form

See Also

ColumnAdd Method, page 224, **SetWidth Method**, page 242, and **Grid Property**, page 231.

GridRows Property

This property gets or sets the number of rows defined on a grid; it is read or write.

Syntax

```
object.GridRows = value
```

or

```
result = object.GridRows
```

Remarks

This property is initially populated by the Load method for the class. Typically, Visual Basic maintenance dialogs do not use this property. However, the DropDownGrid method for the class uses the GridRows property for another instance of the class.

Example

```
Product.Grid.Grid.Height= _
     (ProductGrid.Grid.RowHeight*ProductGrid.GridRows)+30
```

See Also

Load Method, page 237, and **DropDownGrid Method**, page 228.

HighestUsedRow Method

This method returns the number of the grid row that was last populated.

Syntax

```
object.HighestUsedRow
```

Remarks

This method assumes a row is populated if there is a value in any column in the grid row. If the grid is completely empty, this method returns 0.

Note: This method uses business data type (BDT) logic to determine whether a cell is populated. If you add custom BDTs, also customize the logic in this method.

Example

```
LastRow = ProductsGrid.HighestUsedRow
```

Data Type

Integer

HighlightCurrentCell Method

This method places a black highlight rectangle around the current grid cell if it is non-modifiable.

Syntax

object.HighlightCurrentCell()

Remarks

This method is called in the GotFocus event code for a grid. It has no effect on cells that use the FloatingEditor marquee style (cells that users can type in).

Example

```
Private Sub grd_ORD_NcstOrderHasLines_GotFocus()
    Dim NcstOrderHasLinesRow As Integer
    NcstOrderHasLinesGrid.HighlightCurrentCell
```

hwnd Property

This property gets or sets the handle for the True DBGrid; it is read or write.

Syntax

object.hwnd = value

or

result = *object*.hwnd

Remarks

This property provides the ErrorTip logic with polymorphic behavior. The GetControlCoordinates function in CSTUtils.bas uses the hwnd property of the object stored in an ObjectError object to retrieve the absolute screen coordinates for the control. An ObjectError object stores a reference to the TrueGridClass object rather than a True DBGrid. This property returns the hwnd property of the underlying True DBGrid associated with the TrueGridClass object.

Example

```
‘Get the top left and bottom right coordinates of the linked
‘TrueGridClass stored in ErrControl.
lresponse=GetWindowRect(hwnd:=DispObjectError.ErrControl.hwnd, _
     lpRect:=CtrlRect)
ASSERT GetControlCoordinates<>0,_
      “Error in GetWindowRect call for DispObjectError.ErrControl.hwnd”
```

See Also

Object Error Classes, page 247.

InsertGridRow Method

This method inserts a row of blank data before the currently selected row in a grid.

Syntax

object.InsertGridRow()

Remarks

When this method is invoked, the number of grid rows available to the grid is not increased by one. Instead, the data for the selected row and all subsequent rows is moved down one and a new row of data is inserted. This method works with the GridData property.

Example

```
NcstOrderHasLinesGrid.InsertGridRow
```

See Also

DeleteGridRow Method, page 228, and **GridData Property**, page 232.

Load Method

This method initializes an instance of the TrueGridClass class.

Syntax

object.Load *GridRows*, *Headings*, *RowCounter*

Part	Required	Data Type	Description
GridRows	yes	Integer	Number of rows in the grid.
Headings	no	Boolean	Indicates whether the grid has column headings. The default is True (the grid has column headings).
RowCounter	no	Boolean	Indicates whether the grid has a counter column. The default is True (the grid has a counter column).

Remarks

The Load method is called when the form is initially loaded. It must be called before any ColumnAdd methods can be called for the TrueGridClass class.

Example

```
Private Sub LoadNcstOrderHasLinesGrid()
    With NcstOrderHasLinesGrid
        Set .Grid = grd_ORD_NcstOrderHasLines
        Set .GridForm = Me
        .Load MAX_NCSTORDERHASLINES_ROWS
        .ColumnAdd "Line/Number", BDT_NUMERIC, "N2"
        .ColumnAdd "Product/Group", BDT_ALPHA, "A2"
        .ColumnAdd "Quantity", BDT_NUMERIC, "P9"
        .ColumnAdd "Unit/Cost", BDT_CURRENCY, "P7.2"
        .ColumnAdd "Total/Cost", BDT_CURRENCY, "P9.2", _
                    Modifiable:=False
        .SetWidth True, True
    End With
End Sub
```

See Also

ColumnAdd Method, page 224, **Grid Property**, page 231, and **GridForm Property**, page 233.

LostFocus Method

This method removes any marquee cell highlighting from a grid cell.

Syntax

object.LostFocus

Remarks

This method is typically called from the LostFocus event for a grid on a form. It removes “focus” highlighting from the cell that has just lost focus.

Example

```
Private Sub grd_ORD_NcstOrderHasLines_LostFocus()
    NcstOrderHasLinesGrid.LostFocus
    CSTUtils.ErrorTip.HideErrorTip
End Sub
```

Name Property

This property contains the name of the True DBGrid control associated with this instance of the TrueGridClass class.

Syntax

object.Name

Remarks

This property is read only. It is included as part of the TrueGridClass class to provide polymorphic behavior for the Control property of the ObjectError class.

Example

```
colErrors.Add Item:= MyError, _
              Key:= (pErrControl.Name & Str(pErrColumn) & _
                    Str(pMsgType) & Str(pIndex1) & Str(pIndex2) & _
                    Str(pIndex3) & Str(pMsgNr))
```

Data Type

String

NotInBounds Method

This method returns a Boolean value indicating whether the supplied grid row value is within the bounds of rows that are currently available on the grid.

Syntax

object.NotInBounds(*GridRow*)

Part	Required	Data Type	Description
GridRow	yes	Integer	Grid row value to be validated.

Remarks

To ensure that a row index is valid, use this method before manipulating grid data. Typically, you call this method after calling the CurrentRow method.

Example

```
CurrRow = NcstOrderHasLinesGrid.CurrentRow
If NcstOrderHasLinesGrid.NotInBounds(CurrRow) Then
    ValidateGridAction = False
    Exit Function
End If
```

Data Type

Boolean

See Also

CurrentRow Method, page 227.

Parent Property

This property contains a reference to the form for the True DBGrid control associated with this instance of the TrueGridClass class.

Syntax

object.Parent

Remarks

This property is read only. It is included as part of the TrueGridClass class to provide polymorphic behavior for the DisplayErrorTip subroutine in CSTUtils.bas.

Example

```
Dim ParentForm As Form
…
Set ParentForm = DispObjectError.ErrControl.Parent
```

Data Type

Object

ReadGridData Method

This method returns a variant variable containing a 2-dimensional array of grid data from the GridData property.

Syntax

object.ReadGridData()

Remarks

Object data with more than two dimensions cannot be contained in the GridData property, which only has two dimensions. This method and the WriteGridData method read and write data between variant arrays (up to four dimensions) and the GridData property. If a maintenance dialog contains nested grids (grids that represent data with more than two dimensions), it uses this method.

Example

```
If (LastProductRow + 1) <> ProductsRow Then
    DistributionsArray((LastProductRow + 1)) = _
                    DistributionsGrid.ReadGridData
    DistributionsGrid.WriteGridData DistributionsArray(ProductsRow)
    grd_ORD_Distributions.Refresh
    ChangeDistributionsLabel
End If
```

Data Type

Variant

See Also

WriteGridData Method, page 245, and **GridData Property**, page 232.

RowColChange Method

This method sets the marquee style for a grid based on the column configuration information established when the ColumnAdd method was invoked.

Syntax

object.RowColChange *Col*

Part	Required	Data Type	Description
Col	yes	Integer	Current column.

Remarks

This method is typically invoked from the RowColChange event code for the grid on a form.

Example

```
Private Sub grd_ORD_Products_RowColChange(LastRow As Variant, _
                    ByVal LastCol As Integer)
    Dim NcstOrderHasLinesRow As Integer
    If Not IsNumeric(LastRow) Then LastRow = 0
    NcstOrderHasLinesRow = NcstOrderHasLinesGrid.CurrentRow
    If NcstOrderHasLinesGrid.NotInBounds(NcstOrderHasLinesRow) Then
        Exit Sub
    End If
    NcstOrderHasLinesGrid.RowColChange grd_ORD_NcstOrderHasLines.Col
```

SetWidth Method

This method resizes the width of a grid to display as many grid columns as possible.

Syntax

object.SetWidth *ShrinkHeightAsRequired*, *RecordRequiredSizeMsg*

Part	Required	Data Type	Description
ShrinkHeight AsRequired	no	Boolean	Indicates whether the method decreases the height of the grid, if applicable. The default is False.
RecordRequired SizeMsg	no	Boolean	Indicates whether the method records grid sizing information. The default is False.

Remarks

The method ensures that the grid width never causes the grid control to be clipped by the right edge of the form. The method determines whether horizontal or vertical scroll bars are required, based on whether all columns or all rows on the grid are visible.

If the RecordRequiredSizeMsg argument is True, you can display grid sizing information for the grid by making a subsequent call to the DisplayGridSizingInfo routine.

Example

```
Private Sub LoadNcstOrderHasLinesGrid()
    With NcstOrderHasLinesGrid
        Set .Grid = grd_ORD_NcstOrderHasLines
        Set .GridForm = Me
        .Load MAX_NCSTORDERHASLINES_ROWS
        .ColumnAdd "Line/Number", BDT_NUMERIC, "N2"
        .ColumnAdd "Product/Group", BDT_ALPHA, "A2"
        .ColumnAdd "Quantity", BDT_NUMERIC, "P9"
        .ColumnAdd "Unit/Cost", BDT_CURRENCY, "P7.2"
        .ColumnAdd "Total/Cost", BDT_CURRENCY, "P9.2", _
                    Modifiable:=False
        .SetWidth True, True
    End With
End Sub
```

See Also

ColumnAdd Method, page 224, **Grid Property**, page 231, and **GridForm Property**, page 233.

UnboundReadData Method

This method transfers data from the GridData property to the internal row buffer for a True DBGrid.

Syntax

object.UnboundReadData *RowBuf*, *StartLocation*, *ReadPriorRows*

Part	Required	Data Type	Description
RowBuf	yes	RowBuf	Internal True DBGrid object.
StartLocation	yes	variant	Bookmark value that uniquely identifies a grid row.
ReadPriorRows	yes	Boolean	Direction in which rows from the GridData property are read.

Remarks

This method should only be called from the UnboundReadData event code for a grid on the form. Use the arguments raised in the event code as the arguments for this method — they are named identically.

Note: For information on using a grid in "Unbound" mode, see the online help.

Example

```
Private Sub grd_ORD_Products_UnboundReadData(ByVal RowBuf As _
                    RowBuffer, StartLocation As Variant, _
                    ByVal ReadPriorRows As Boolean)
    ProductsGrid.UnboundReadData RowBuf, StartLocation, ReadPriorRows
End Sub
```

See Also

UnboundWriteData Method, page 244, and **GridData Property**, page 232.

UnboundWriteData Method

This method transfers data from the internal row buffer for a True DBGrid to the GridData property.

Syntax

object.UnboundWriteData (*RowBuf*, *WriteLocation*)

Part	Required	Data Type	Description
RowBuf	yes	RowBuf	Internal True DBGrid object.
WriteLocation	yes	variant	Bookmark value that uniquely identifies a grid row.

Remarks

This method should only be called from the UnboundWriteData event code for a grid on a form. Use the arguments raised in the event code as the arguments for this method — they are named identically.

Note: For information on using a grid in "Unbound" mode, see the online help.

Example

```
Private Sub grd_ORD_Products_UnboundWriteData(ByVal RowBuf _
                    As RowBuffer, WriteLocation As Variant)
    ProductsGrid.UnboundWriteData RowBuf, WriteLocation
End Sub
```

See Also

UnboundReadData Method, page 243, and **GridData Property**, page 232.

WriteGridData Method

This method writes a variant variable containing a 2-dimensional array of grid data to the GridData property.

Syntax

object.WriteGridData *Arr*

Remarks

Object data that has more than two dimensions cannot be contained in the GridData property, which only has two dimensions. The ReadGridData method and this method read and write data between variant arrays (up to four dimensions) and the GridData property.

Example

```
If (LastProductRow + 1) <> ProductsRow Then
    DistributionsArray((LastProductRow + 1)) = _
                    DistributionsGrid.ReadGridData
    DistributionsGrid.WriteGridData DistributionsArray(ProductsRow)
    grd_ORD_Distributions.Refresh
    ChangeDistributionsLabel
End If
```

See Also

ReadGridData Method, page 240, and **GridData Property**, page 232.

OBJECT ERROR CLASSES

This chapter describes the object error classes, ObjectError and ObjectErrors, as well as their methods and properties. These classes provide a standard mechanism for storing and retrieving error information pertaining to a specific GUI control.

The following topics are covered:

For related information, see:

ObjectError Class

This class stores an error associated with a specified GUI control. Typically, you manipulate ObjectError objects through the methods of an ObjectErrors object.

The ObjectError class does not use any public methods; it has the following properties:

ErrColumn Property, page 248	**MsgNr Property**, page 250
ErrControl Property, page 249	**MsgType Property**, page 251
Index1, Index2, and Index3 Properties, page 249	**RegBackColor and RegForeColor Properties**, page 251
Msg Property, page 250	**SoundFile Property**, page 252

ErrColumn Property

This property identifies the column within a grid control that is linked to this object error; it is read or write.

Syntax

object.ErrColumn = *value*

or

result = *object*.ErrColumn

Remarks

This property is only applicable to ObjectError objects that are linked to TrueGridClass objects.

Example

```
Dim ColIndex as Integer
ColIndex = ObjError.ErrColumn
```

Data Type

Integer

See Also

TrueGridClass Class, page 222.

ErrControl Property

This property identifies a GUI control that the ObjectError object is attached to; it is read or write.

Syntax

```
Set object.ErrControl = value
```

or

```
Set result = object.ErrControl
```

Remarks

This property can reference either a GUI control or a TrueGridClass object. Before attempting to invoke any object-specific methods or properties, test for the type of object this property is referencing.

Example

```
If CurrControl Is MyError.ErrControl Then
     DisplayErroTip MyError
End If
```

Data Type

Object

Index1, Index2, and Index3 Properties

These properties help relate an ObjectError object to a specified occurrence of array data; it is read or write.

Syntax

```
object.Index1 = value
```

or

```
result = object.Index1
```

Remarks

These properties are used for array data only. Typically, the object associated with an ObjectError object that uses the Index properties is a TrueGridClass object or a GUI control that is part of a control array.

Example

```
If TypeOf ObjError.ErrControl Is TrueGridClass Then
     CurrRow = ObjError.Index1
End If
```

Data Type

Integer

Msg Property

This property contains an error message that describes the ObjectError object attached to a GUI control; it is read or write.

Syntax

object.Msg = *value*

or

result = *object*.Msg

Data Type

String

Example

```
frmErrorTip.lblMessage.Caption = ObjError.Msg
```

MsgNr Property

This property contains an error number that, in combination with the MsgType property, uniquely identifies a server error.

Syntax

object.MsgNr = *value*

or

result = *object*.MsgNr

Data Type

Long

Example

```
ObjError.MsgNr = InternalObject.Msg.Field(“MSG-NR”)
```

MsgType Property

This property identifies the source of an ObjectError object; it is read or write.

Syntax

object.MsgType = *value*

or

result = *object*.MsgType

Remarks

An enumerated list of constants is supplied in CSTObjectConst.bas (prefixed by ERROR_SOURCE). Use these constants to uniquely identify the message type.

Data Type

Integer

Example

```
ObjError.MsgType = ERROR_SOURCE_SERVER
```

RegBackColor and RegForeColor Properties

These properties contain the (regular, non-error) RGB background and foreground colors for a GUI control attached to an ObjectError object; it is read or write.

Syntax

object.RegBackColor = *value*

or

result = *object*.RegBackColor

Remarks

These properties are set when an ObjectError object is added to the ObjectErrors object collection through the Add method of the ObjectErrors object. The ObjectErrors object controls the BackColor and ForeColor properties of the related GUI control (based on whether errors are attached to the GUI control).

Example

```
MyError.RegForeColor = pRegForeColor
```

Data Type

Long

See Also

cstDisplayErrorTip Subroutine, page 376, and **HideErrorTip Subroutine**, page 386.

SoundFile Property

This property contains the name of a .wav file.

Remarks

You can use this file name to provide audio error messages in your application. To support audio error messages, you must provide custom code.

Data Type

String

See Also

Add Method, page 253.

ObjectErrors Class

This class provides a unified storage mechanism for business errors associated with a specified GUI control. An ObjectErrors object can house a collection of individual ObjectError objects. Each browse and maintenance form declares one instance of the ObjectErrors class and uses the methods and properties associated with this class to add, remove, and manipulate an individual ObjectError object.

The ObjectErrors class does not use any public properties; it has the following methods:

Add Method, page 253	**Item Method**, page 257
AlterControlErrorIndices Method, page 254	**Remove Method**, page 259
Count Method, page 255	**RemoveByIndex Method**, page 261

Add Method

This method adds a new ObjectError object to the ObjectErrors class. An ObjectError object is created based on the input parameters for this method.

Syntax

```
object.Add ErrControl, Msg, MsgType, ErrColumn, Index1, Index2, _
                                                  Index3, MsgNr
```

Part	Required	Data Type	Description
ErrControl	yes	object	Reference to a control or TrueGridClass object.
Msg	yes	string	Message describing the nature of the error for the ObjectError object.
MsgType	yes	Integer	Indicates whether the error was a result of local or remote validation. Use the LOCAL_VALIDATION or REMOTE_VALIDATION constant.
ErrColumn	no	Integer	Column (if the ObjectError object represents a column on a grid).
Index1	no	Integer	Unique occurrence of an array field.
Index2	no	Integer	Unique occurrence of an array field.

Part	Required	Data Type	Description (continued)
Index3	no	Integer	Unique occurrence of an array field.
MsgNr	no	long	Unique number identifying an error that originated on the server.

Remarks

If you supply an index value for a dimension (for example, Index3), you must supply an index value for all lower level dimensions (for example, Index1 and Index2). Index values should be one-based because they reference one-based arrays.

For an example of calls to this method, refer to the SetObjectError routine in CSTUtils.bas.

Example

```
p_ErrForm.ObjErrors.Add p_ErrControl, p_ErrMsg, p_ErrMsgType, _
         p_ErrColumn, p_Index1, p_Index2, p_Index3, p_ErrMsgNr
```

AlterControlErrorIndices Method

This method adjusts the index values of ObjectError objects that are associated with a specified GUI control.

Syntax

object.AlterControlErrorIndices *ErrControl*, *BaseLine*, *Rank*, *Offset*

Part	Required	Data Type	Description
ErrControl	yes	object	Reference to a control or TrueGridClass object that is associated with one or more ObjectError objects.
BaseLine	yes	Integer	ObjectError objects to be adjusted based on the index value for the specified Rank.
Rank	yes	Integer	Index value considered. Valid values are 1, 2, or 3. Based on this value, the method considers the Index1, Index2, or Index3 values for an ObjectError object.
Offset	yes	Integer	Amount the specified Index is adjusted (positive or negative).

Remarks

Note: The Rank value refers to the Index value with the matching suffix. For example, if the Rank is 1, the Rank Index is Index1.

The method iterates through all ObjectError objects in the class. It alters the Rank index if the ErrControl parameter matches the ErrControl parameter in the ObjectError object and the Rank index parameter is greater than the Baseline parameter.

This method keeps the index values in an ObjectErrors object synchronized with the index values for a grid on a form. To alter index values on a grid, use the InsertGridRow or DeleteGridRow methods for the TrueGridClass class.

Example

In this example, all ObjectError objects associated with the NcstOrderHasLinesGrid object that have an Index1 property value greater than 4 will have their Index1 property decremented by one.

```
ObjErrors.AlterControlErrorIndices NcstOrderHasLinesGrid, _
          4, 1, -1
```

See Also

InsertGridRow Method, page 236, and **DeleteGridRow Method**, page 228.

Count Method

This method returns an integer value indicating the number of ObjectError objects associated with the ObjectErrors object matching the specified input parameters.

Syntax

```
result = object.Count(ErrControl, MsgType, ErrColumn, Index1, Index2, _
                                                       Index3, MsgNr)
```

Part	Required	Data Type	Description
ErrControl	no	object	Reference to a control associated with an ObjectError object.
MsgType	no	Integer	Indicates whether the error was a result of local or remote validation. Use the LOCAL_VALIDATION or REMOTE_VALIDATION constant.

Part	Required	Data Type	Description (continued)
ErrColumn	no	Integer	Column (if the ObjectError object represents a column on a grid).
Index1	no	Integer	Unique occurrence of an array field.
Index2	no	Integer	Unique occurrence of an array field.
Index3	no	Integer	Unique occurrence of an array field.
MsgNr	no	long	Unique number identifying an error that originated on the server.

Remarks

This method uses optional input parameters as filters to determine what values are returned. The following table lists the eight modes in which this method operates:

Note: If you do not pass the Index1, Index2, or Index3 values to this method, they default to zero.

Passed Parameters	Missing Parameters	Returns Count of
	ErrControl, ErrColumn, MsgType, MsgNr	All errors on the form.
ErrControl	ErrColumn, MsgType, MsgNr	All errors for a specified control; grid columns are not considered separately.
ErrControl, ErrColumn	MsgType, MsgNr	All errors for a specified control; grid columns are considered separately.
ErrControl, MsgType	ErrColumn, MsgNr	All local or remote errors for a specified control; grid columns are not considered separately.
ErrControl, MsgType, ErrColumn	MsgNr	All local or remote errors for a specified control; grid columns are considered separately.
MsgType	ErrControl	All local or remote errors for this form.

Passed Parameters	Missing Parameters	Returns Count of (continued)
ErrControl, MsgType, MsgNr	ErrColumn	Specified host error for this control; grid columns are not considered separately.
ErrControl, MsgType, MsgNr, ErrColumn		Specified host error for this control; grid columns are considered separately.

Example

```
If ObjErrors.Count(pMsgType:=LOCAL_VALIDATION) > 0 Then
    .Enabled = False
    .DisabledReason = csterrLocalErrorsMsg
End If
```

Data Type

Integer

Item Method

This method returns a reference to an ObjectError object for the specified input parameters.

Syntax

```
set result = object.Item(ErrControl, MsgType, ErrColumn, Index1, _
                         Index2, Index3, MsgNr) As ObjectError
```

Part	Required	Data Type	Description
ErrControl	yes	object	Reference to a control associated with an ObjectError object.
MsgType	no	Integer	Indicates whether the error was a result of local or remote validation. Use the LOCAL_VALIDATION or REMOTE_VALIDATION constant.
ErrColumn	no	Integer	Column (if the error represents a column on a grid).
Index1	no	Integer	Unique occurrence of an array field.
Index2	no	Integer	Unique occurrence of an array field.

Part	Required	Data Type	Description (continued)
Index3	no	Integer	Unique occurrence of an array field.
MsgNr	no	long	Unique number identifying an error that originated on the server.

Remarks

This method uses optional input parameters as filters to return the first ObjectError object that matches the specified input parameters. The following table lists the four different modes in which this method operates:

Note: If you do not pass the ErrColumn, Index1, Index2, or Index3 values to this method, they default to a zero.

Passed Parameters	Missing Parameters	Returns First
	MsgNr, MsgType	ObjectError object containing ErrControl, ErrColumn, and Index values matching the specified input parameters.
MsgType	MsgNr	ObjectError object containing ErrControl, ErrColumn, Index, and MsgType values matching the specified input parameters.
MsgNr	MsgType	ObjectError object containing ErrControl, ErrColumn, Index, and MsgNr values matching the specified input parameters.
MsgNr, MsgType		ObjectError object containing ErrControl, ErrColumn, Index, MsgType, and MsgNr values matching the specified input parameters.

Example

```
Set MyError = CurrForm.ObjErrors.Item(CurrControl, _
                                      ErrColumn:=ErrColumn, _
                                      Index1:=Index1, _
                                      Index2:=Index2, _
                                      Index3:=Index3)
If Not MyError Is Nothing Then
```

Data Type

ObjectError

Remove Method

This method removes any ObjectError objects matching the specified input parameters from the ObjectErrors object; it returns the number of ObjectError objects removed.

Syntax

```
result = object.Remove ErrControl, MsgType, ErrColumn, Index1, Index2, _
                                                    Index3, MsgNr As Integer
```

Part	Required	Data Type	Description
ErrControl	no	object	Reference to a control associated with an ObjectError object.
MsgType	no	Integer	Indicates whether the error was a result of local or remote validation. Use the LOCAL_VALIDATION or REMOTE_VALIDATION constant.
ErrColumn	no	Integer	Column (if the error represents a column on a grid).
Index1	no	Integer	Unique occurrence of an array field.
Index2	no	Integer	Unique occurrence of an array field.
Index3	no	Integer	Unique occurrence of an array field.
MsgNr	no	long	Unique number identifying an error that originated on the server.

Remarks

This method uses optional input parameters as filters to determine which ObjectError objects to remove. The following table lists the modes in which this method operates:

Note: If you do not pass the Index1, Index2, or Index3 values to this method, they default to zero.

Passed Parameters	Missing Parameters	Returns Count of
	ErrControl, ErrColumn, MsgType, MsgNr	All errors on the form.
ErrControl	ErrColumn, MsgType, MsgNr	All errors for a specified control; grid columns are not considered separately.
ErrControl, ErrColumn	MsgType, MsgNr	All errors for a specified control; grid columns are considered separately.
ErrControl, MsgType	ErrColumn, MsgNr	All local or remote errors for a specified control; grid columns are not considered separately.
ErrControl, MsgType, ErrColumn	MsgNr	All local or remote errors for a specified control; grid columns are considered separately.
MsgType	ErrControl	All local or remote errors for this form.
ErrControl, MsgType, MsgNr	ErrColumn	Specified host error for this control; grid columns are not considered separately.
ErrControl, MsgType, MsgNr, ErrColumn		Specified host error for this control; grid columns are considered separately.

Example

```
ErrorCount = CurrForm.ObjErrors.Remove(ErrControl:=CurrControl, _
                                       ErrColumn:=ErrColumn, _
                                       Index1:=Index1, _
                                       Index2:=Index2, _
                                       Index3:=Index3)
```

Data Type

Integer

RemoveByIndex Method

This method removes ObjectError objects from an ObjectErrors object based on a specified control reference and indices. It returns an integer indicating the number of ObjectError objects removed.

Syntax

object.RemoveByIndex(*ErrControl*, *Index1*, *Index2*, *Index3*) As Integer

Part	Required	Data Type	Description
ErrControl	yes	Object	Reference to a control associated with an ObjectError object.
Index1	no	Integer	Unique occurrence of an array field.
Index2	no	Integer	Unique occurrence of an array field.
Index3	no	Integer	Unique occurrence of an array field.

Remarks

When you delete an entire row from a grid, use this version of the Remove method to remove ObjectError objects for the grid controls. This method removes all ObjectError objects matching the input parameters for ErrControl and Index1. If Index2 and Index3 are specified, this method restricts the match criteria to ObjectError objects that also match these additional parameters.

Example

The following example removes all ObjectError objects from “MyGridControl” that have a Rank Index1 value of 1, regardless of their Index2 or Index3 values:

```
ErrCount = ObjErrors.RemoveByIndex(MyGridControl, Index1:=1)
```

Data Type

Integer

See Also

DeleteGridRow Method, page 228.

OBJECT FACTORY

This chapter describes the object factory module, which identifies and instantiates all objects and methods in an application. It also describes the Open dialog.

The following topics are covered:

Object Factory Module

The object factory module keeps the client portion of an application aware of all its business objects and the actions associated with them. This module:

- Identifies all objects and methods in an application
- Instantiates objects upon request for use by other components of an application

Constants Used

To identify business objects in an application, the object factory uses string constants. These constants are derived from the names on the database tables used by the business objects. In the following example, the GetBrowser() function receives the name of a constant as input and returns a BrowseManager object:

```
Dim BrMgr As BrowseManager
Set BrMgr = GetBrowser("EMPLOYEES")
```

Application components that use the public functions and subroutines for the object factory module must specify the correct database table names.

Functions and Subroutines Used

The object factory has the following functions and subroutine:

BrowserExists Function, page 264	**GetBrowser Function**, page 265
CreateForm Function, page 265	**InitializeOpenDialog Subroutine**, page 266

BrowserExists Function

This function confirms whether a Browse object associated with a database table name exists; it returns either True or False.

Syntax

```
BrowserExists(TableName) As Boolean
```

Part	Required	Data Type	Description
TableName	yes	string	Name of the database table for which the specified Browse object is implemented.

CreateForm Function

This function creates a form to support an action for a business object (browse or maintenance); it returns a reference to the form.

Syntax

```
CreateForm(formID) As Form
```

Part	Required	Data Type	Description
formID	yes	variant	Unique identification for an object action that requires a form to be displayed. This parameter must map to a Case in the SelectCase statement for this function.

Remarks

This function is called by Open.frm in the Construct Spectrum client framework. The formID value is linked to the object action by the InitializeOpenDialog() subroutine when it builds the OpenObjects class. Before you can use the CreateForm function, you must first call the InitializeOpenDialog subroutine to build the OpenObjects class.

See Also

OpenObject Class, page 269.

GetBrowser Function

This function:

- Creates a Browse object for the specified database table.
  The Browse object then creates and initializes a BrowseBase object.
- Creates a BrowseManager object.
- Initializes the BrowseManager object with a reference to the BrowseBase object.
- Returns a reference to the BrowseManager object.

Syntax

```
GetBrowser(TableName) As BrowseManager
```

Part	Required	Data Type	Description
TableName	yes	string	Name of the database table for which the specified Browse object is implemented.

See Also

Browse Classes, page 49.

InitializeOpenDialog Subroutine

This subroutine creates a list of the business objects used by an application and the actions they support; the list is available to the entire application.

Syntax

```
InitializeOpenDialog()
```

Remarks

This subroutine is called by Open.frm in the Construct Spectrum client framework.

See Also

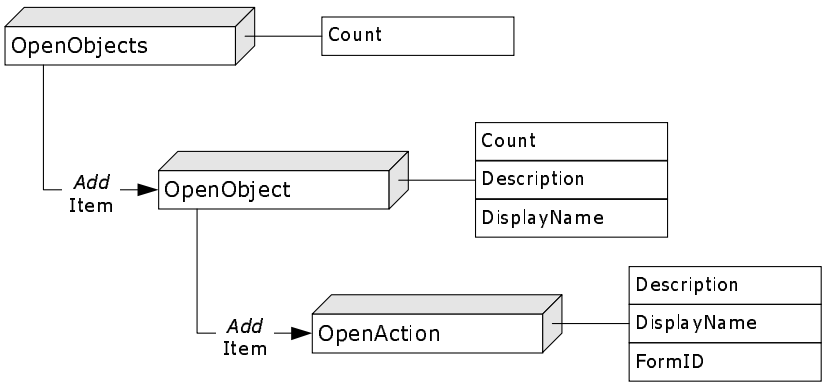
Open Dialog Support, page 267.

Open Dialog Support

The Construct Spectrum client framework includes three classes that provide support for the Open dialog:

- OpenAction class
- OpenObject class
- OpenObjects class

The following diagram shows the structure of the Open dialog classes:



Structure of the Open Dialog Classes

Objects in these classes are typically used as follows:

```
Add <business object 1> to OpenObjects returning a new OpenObject
    Add <action 1> to OpenObject
    Add <action 2> to OpenObject
    Add <action 3> to OpenObject
    Add <action 4> to OpenObject

Add <business object 2> to OpenObjects returning a new OpenObject
    Add <action 1> to OpenObject
    Add <action 2> to OpenObject
    Add <action 3> to OpenObject
...
```

The following sections describe these classes and their properties and methods.

OpenAction Class

This class returns information about a single action defined for a business object registered to the Open dialog.

The OpenAction class has the following properties:

Description Property, page 268	**FormID Property**, page 269
DisplayName Property, page 268	

Description Property

This property returns the description of an action defined for a business object registered to the Open dialog.

Syntax

```
result = object.Description
```

Part	Required	Data Type	Description
object	yes	OpenAction	Name of the OpenAction object.
result		string	Description of the specified OpenAction object.

DisplayName Property

This property returns the display name for a business object action on the Open dialog.

Syntax

```
result = object.DisplayName
```

Part	Required	Data Type	Description
object	yes	OpenAction	Name of the OpenAction object.
result		string	Display name for the specified OpenAction object.

FormID Property

This property returns the form ID associated with a business object action on the Open dialog.

Syntax

```
result = object.FormID
```

Part	Required	Data Type	Description
object	yes	OpenAction	Name of the OpenAction object.
result		variant	Form ID associated with the specified OpenAction object.

OpenObject Class

This class maintains information about a single business object registered to the Open dialog.

The OpenObject class has the following methods and properties:

Add Method, page 269	**DisplayName Property**, page 271
Count Property, page 270	**Item Property**, page 271
Description Property, page 271	

Add Method

This method adds an action to a specified business object.

Syntax

```
Set result = object.Add(DisplayName, Description, FormID)
```

Part	Required	Data Type	Description
object	yes	OpenObject	Name of the OpenObject object.
DisplayName	yes	string	Name displayed in the Actions list box on the Open dialog.

Part	Required	Data Type	Description (continued)
Description	yes	string	Description displayed at the bottom of the Open dialog when the action is selected.
FormID	yes	variant	Any scalar variant value except an object reference.
result		OpenAction	Instance of the OpenAction class used to read the DisplayName, Description, and FormID parameters.

Remarks

- The DisplayName value must be unique for all actions added to the business object.
- You should associate a form identification value (FormID) with each business object and action combination.
- When the user selects an object and action combination on the Open dialog, the Open dialog passes the FormID value to the CreateForm function in the object factory. The object factory creates a form and returns it to the Open dialog. The Open dialog then displays the form.

See Also

Count Property, page 270, and **Item Property**, page 271.

Count Property

This property returns the number of actions defined for a business object.

Syntax

```
result = object.Count
```

Part	Required	Data Type	Description
object	yes	OpenObject	Name of the OpenObject object.
result		long	Number of actions defined for the specified business object.

See Also

Add Method, page 269, and **Item Property**, page 271.

Description Property

This property returns the description defined for a business object.

Syntax

```
result = object.Description
```

Part	Required	Data Type	Description
object	yes	OpenObject	Name of the OpenObject object.
result		string	Description defined for the specified business object.

DisplayName Property

This property returns the name displayed in the Objects list box on the Open dialog.

Syntax

```
result = object.DisplayName
```

Part	Required	Data Type	Description
object	yes	OpenObject	Name of the OpenObject object.
result		string	Display name for the specified business object.

See Also

Count Property, page 270, **Description Property**, page 271, and **Item Property**, page 271.

Item Property

This property returns a single OpenAction object from the OpenObjects class.

Syntax

```
Set result = object.Item(Index)
```

Part	Required	Data Type	Description
object	yes	OpenObjects	Name of the OpenObjects object.
Index	yes	variant	Either a numeric index or the display name for an action.
result		OpenAction	OpenAction object associated with the specified Index value.

Data Type

If the specified Index value does not exist in the class, this property returns nothing.

See Also

Add Method, page 269, and **Count Property**, page 270.

OpenObjects Class

This class registers business objects displayed in the Objects list box on the Open dialog.

The OpenObjects class has the following methods and properties:

Add Method, page 269	**Item Property**, page 271
Count Property, page 270	

Add Method

This method adds a business object to the Open dialog.

Syntax

```
Set result = object.Add(DisplayName, Description)
```

Part	Required	Data Type	Description
object	yes	OpenObjects	Name of the OpenObjects object.
DisplayName	yes	string	Business object name displayed in the Objects list box on the Open dialog.
Description	yes	string	Description displayed at the bottom of the Open dialog when the business object is selected.
result		OpenObject	Instance of OpenObject class used to add actions to the Open dialog.

Remarks

The DisplayName value must be unique for all business objects added to the OpenObjects class. To add actions to the object, use the object returned by this method.

Count Property

This property returns the number of business objects defined for the Open dialog.

Syntax

```
result = object.Count
```

Part	Required	Data Type	Description
object	yes	OpenObjects	Name of the OpenObjects object.
result		long	Number of business objects defined for the specified Open dialog.

Item Property

This property returns a single OpenObject object from the OpenObjects class.

Syntax

```
Set result = object.Item(Index)
```

Part	Required	Data Type	Description
object	yes	OpenObjects	Name of the OpenObjects object.
Index	yes	variant	Either a numeric index or the DisplayName value for a business object.
result		OpenObject	OpenObject object associated with the specified Index value.

Data Type

If the Index value does not exist in the specified OpenObjects class, nothing is returned.

PAGE HANDLER CLASSES

This chapter describes the page handler classes and their methods and properties. These client framework components are described in the following sections.

The following topics are covered:

ABO Class

This class handles communication between your Construct Spectrum web application and an instance of an ABO object. It encapsulates the validation of data and the execution of methods and properties.

Methods and Properties

The ABO class has the following methods and properties:

ABO Object Property, page 276	**BDT Property**, page 277
Error Property, page 277	**ErrorCount Property**, page 278
Field Property, page 278	**LogicalFormatBDT Property**, page 279
SelectContents Property, page 279	**ClearErrorCache Property**, page 280
GetField Method, page 280	**Init Method**, page 281
InvokeMethod Method, page 281	**SetField Method**, page 282
UpdateFromRequest Method, page 282	

ABO Object Property

This property sets a reference to an ABO object.

Syntax

`Set object.ABOObject = ABO`

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.
ABO	yes	object	Reference to an ABO object.

BDT Property

This property changes the default BDT used for the specified field. The BDT is used to validate and format data.

Syntax

```
object.BDT(FieldName) = BDT
```

Part	Required	Data Type	Description
object	yes	ABOInterface	BDTController object with which the BDT routines are registered.
FieldName	yes	string	Field name (property) in the ABO object.
BDT	yes	string	BDT name.

Error Property

This property returns information about validation errors for a read-only field.

Syntax

```
object.Error (Key) = BDTError
```

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.
Key	yes	variant	Numeric index or field name.
BDTError	yes	BDTError_t	Returns a user-defined type containing a field name, error message, and error value.

ErrorCount Property

This property returns the number of validation errors for a read-only field.

Syntax

object.ErrorCount

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.

Field Property

This property returns the value for a specified field.

Syntax

object.Field (*FieldSpec*)

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.
FieldSpec	yes	string	Name of a field in an ABO object (including indices, if required).

Remarks

If the field contains indices, include them within brackets after the field name. For example:

```
Sval = Object.Field (“CustomerNumber (4)”)
```

See Also

GetField Method, page 280, and **SetField Method**, page 282.

LogicalFormatBDT Property

This property returns the name of a BDT used for a logical format.

Syntax

object.LogicalFormatBDT (*LogicalFormat*) = *BDT*

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.
BDT	yes	string	BDT name.
LogicalFormat	yes	string	Logical format name.

Remarks

An ABO object returns logical formats through the ICSTPropertyInfo interface. By default, the logical formats map directly to BDT names. You can use this property to override the default. For example, to change the Phone logical format to use an alphanumeric BDT, code the following:

```
Object.LogicalFormatBDT (“Phone”) = “Alpha”
```

See Also

BDT Property, page 277.

SelectContents Property

This property returns the name of the BDT used for a logical format. Use this property to add the correct HTML attributes to a combination box (<SELECT>TAG).

Syntax

NewHTML = *object*. SelectContents (*Field*, *CurrentHTML*)

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.
Field	yes	string	ABO property name.
CurrentHTML	yes	string	HTML containing <SELECT> tag.

Remarks

An ABO object returns logical formats through the ICSTPropertyInfo interface. By default, the logical formats map directly to BDT names. You can use this property to override the default. For example, to change the Phone logical format to use an alphanumeric BDT, code the following:

```
Object.LogicalFormatBDT ("Phone") = "Alpha"
```

ClearErrorCache Property

This property resets any cached errors. For persistence between requests, errors are cached in the ASP session object.

Syntax

object.ClearErrorCache

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.

GetField Method

This method returns the value for an ABO object.

Syntax

Sval = *object*.GetField(*Field*, *Indexl*, *Index2*, *Index3*)

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.
Field	yes	string	Name of an ABO object.
Index1	no	integer	Optional index value 1.
Index2	no	integer	Optional index value 2.
Index3	no	integer	Optional index value 3.

Init Method

This method initializes the ABO object with required data.

Syntax

```
object.Init SessionKey, ASPSession, ASPRequest
```

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.
SessionKey	yes	string	Key used to store cached data in the ASP session object.
ASPSession	yes	session	ASP session object.
ASPRequest	yes	request	ASP request object.

InvokeMethod Method

This method executes a method exposed by the ABO object.

Syntax

```
object.InvokeMethod MethodName, Parms ()
```

Part	Required	Data Type	Description
Object	yes	ABOInterface	ABO interface object.
MethodName	yes	string	Name of a public method in an ABO object.
Parms()	no	variant array	Parameters supplied to the method.

Remarks

Internally, this method uses the TypeLibInfo component to call the method.

SetField Method

This method sets the value for an ABO object.

Syntax

```
object.SetField Value, Field, Index1, Index2, Index3
```

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.
Value	yes	variant	Value to set.
Field	yes	string	Name of an ABO object.
Index1	no	integer	Optional index value 1.
Index2	no	integer	Optional index value 2.
Index3	no	integer	Optional index value 3.

UpdateFromRequest Method

This method uses the ASP request object to query information sent from an HTML request.

Syntax

```
Success = object.UpdateFromRequest ()
```

Part	Required	Data Type	Description
object	yes	ABOInterface	ABO interface object.
Success	no	Boolean	Update success.

ICSTPageHandler Class

This class provides a standard interface that is implemented by all page handlers.

Methods and Properties

To interact with page handlers, the Spectrum web application has the following methods and properties:

RequiresLogon Property, page 283	**Title Property**, page 283
Content Property, page 284	**Initialize Method**, page 284
Process Method, page 285	

RequiresLogon Property

This property indicates whether a page handler requires the user to be logged on.

Syntax

```
Value = object.RequiresLogon
```

Part	Required	Data Type	Description
object	yes	ICSTPageHandler	Class that implements the interface.
Value	no	Boolean	

Title Property

This property returns a title for the page handler. Titles are used in the web browser caption and on some framework page handlers, such as browse key selection.

Syntax

```
Value = object.Title
```

Part	Required	Data Type	Description
object	yes	ICSTPageHandler	Class that implements the interface.
Value	no	string	Title.

Content Property

This property returns the HTML contents for the page handler. The page handler then reads and parses one or more HTML templates.

Syntax

```
HTML = object.Content(Content ID)
```

Part	Required	Data Type	Description
object	yes	ICSTPageHandler	Class that implements the interface.
ContentID	no	string	Identifier for the type of contents the page handler returns.
HTML	no	string	HTML displayed in the browser.

Remarks

HTML contents are retrieved from a page handler during the main processing loop for web applications and from the cst:CONTENT replacement tab.

Initialize Method

This method initializes a page handler with the RequestData object containing information about the current request, such ASP objects or current user. Generated ABO page handlers also create an instance of the ABO object they are processing.

Syntax

```
object.Intialize RequestData
```

Part	Required	Data Type	Description
object	yes	ICSTPageHandler	Class that implements the interface.
RequestData	yes	RequestDataObject	Data on the current request.

Process Method

This method performs the processing required before accessing the contents of the page handler. ABO page handlers can update properties and execute methods (depending on the data contained in the request).

Syntax

object.Process

Part	Required	Data Type	Description
object	yes	ICSTPageHandler	Class that implements the interface.

Remarks

The method is always called from the main processing loop for web applications.

RESOURCE CLASSES

This chapter describes the resource classes, Resource and ResourceGroup, which provide functions to help you internationalize your Construct Spectrum application.

The following topics are covered:

Resource Class

This class provides methods to read resources from Resource files. This reduces the effort required to localize an application. The client framework declares and initializes an instance of this class in Startup.bas using the following code:

```
Public Res As New CST.Resource
```

The Resource class has the following methods and properties:

GetResourceGroup Method, page 289	**LocalizeForm Method**, page 293
Language Property, page 289	**Message Method**, page 294
LanguageINIKey Property, page 290	**MessageEx Method**, page 295
LanguageRegistryKey Property, page 291	**ResourceFilePath Property**, page 296
LoadBinaryResource Method, page 292	**SetDefaultMessageGroup Method**, page 297
LoadStringResource Method, page 292	

Setting the Language Code

The Language, LanguageINIKey, and LanguageRegistryKey properties allow you to set the language code used to access Resource files. The Resource class uses the language code as a file name extension to obtain the file name of the Resource file.

You must define a mapping between language codes and user languages. For example, you can use the language codes that Natural uses: 1 for English, 2 for German, etc.

Use the Language property to set the language code. To specify the location to read the language code from (either a .INI file or the Windows registry), use the LanguageINIKey or LanguageRegistryKey property. If you use either of these properties, specify the language code in an external .INI file or the Windows registry (instead of the application) and have the Resource class locate the language code. This feature allows you to share the same language code across many applications. If the language code changes, all other applications that point to the same location automatically use the new value.

The Language, LanguageINIKey, and LanguageRegistryKey properties are mutually exclusive; the Resource class uses the most-recently defined setting. For example, if you set the LanguageINIKey property and then set the Language property, the Resource class uses the Language property setting. However, if you set the LanguageINIKey or LanguageRegistryKey property, you can read the Language property to obtain the external language code value stored at that location.

GetResourceGroup Method

This method creates a ResourceGroup object that returns a list of the resources in a resource group.

Syntax

```
Set result = object.GetResourceGroup(ResourceFile, ResourceGroup)
```

Remarks

If the specified Resource file does not exist, or if the specified resource group does not exist in the Resource file, this method returns Nothing.

Example

```
Dim rg As ResourceGroup
Dim lindex As Long

Set rg = Res.GetResourceGroup("Forms", "frmOpen")
If rg Is Nothing Then
    Print "The resource group does not exist."
Else
    For lindex = 1 To rg.Count
        Print rg.ResourceID(lindex)
    Next
End If
```

Language Property

This property gets or sets the language code; it is read or write.

Syntax

```
object.Language = value
```

or

```
result = object.Language
```

Example

```
Res.Language = "1"
scaption = Res.LoadStringResource("Forms", "frmOpen", "Caption")
```

This example causes the Resource class to access the Forms.1 file and read the Caption resource in the frmOpen resource group.

See Also

LanguageINIKey Property, page 290, **LanguageRegistryKey Property**, page 291, and **Setting the Language Code**, page 288.

LanguageINIKey Property

This property defines a .INI file, section, and key name containing a language code; it is read or write. The Resource class automatically reads the language code from the specified .INI file for every call to the LoadBinaryResource, LoadStringResource, or LocalizeForm method.

Syntax

```
object.LanguageINIKey = value
```

or

```
result = object.LanguageINIKey
```

Remarks

Use this property to specify a valid .INI file, section, and key name; separate each value using a Tab character.

Example

```
.LanguageINIKey = "C:\Windows\CST411.INI" & vbTab & _
                  "Settings" & vbTab & _
                  "Language"
```

See Also

- **Language Property**, page 289
- **LanguageRegistryKey Property**, page 291
- **LoadBinaryResource Method**, page 292
- **LoadStringResource Method**, page 292
- **LocalizeForm Method**, page 293
- **Setting the Language Code**, page 288

LanguageRegistryKey Property

This property defines a Windows registry key containing a language code; it is read or write. The Resource class reads this code from the registry key for every call to the LoadBinaryResource, LoadStringResource, or LocalizeForm method.

Syntax

object.LanguageRegistryKey = *value*

or

result = *object*.LanguageRegistryKey

Remarks

To specify a valid registry key, use this property beginning with one of the following:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
- HKEY_USERS

and ending with a value name.

Example

```
.LanguageRegistryKey = "HKEY_CURRENT_USER\" & _
                       "Software\" & _
                       "Software AG\" & _
                       "CST Frameworks\" & _
                       "Language"
```

See Also

- **Language Property**, page 289
- **LanguageINIKey Property**, page 290
- **LoadBinaryResource Method**, page 292
- **LoadStringResource Method**, page 292
- **LocalizeForm Method**, page 293
- **Setting the Language Code**, page 288

LoadBinaryResource Method

This method loads a resource and returns the resource as a byte array.

Syntax

```
result = object.LoadBinaryResource(ResourceFile, ResourceGroup, _
                                   ResourceID)
```

Remarks

If the specified resource cannot be found, this method returns Null.

See Also

LanguageINIKey Property, page 290, **LanguageRegistryKey Property**, page 291, and **LoadStringResource Method**, page 292.

LoadStringResource Method

This method loads a resource and returns the resource as a string.

Syntax

```
result = object.LoadStringResource(ResourceFile, ResourceGroup, _
                                   ResourceID)
```

Remarks

If the specified resource cannot be found, this method returns an empty string.

See Also

LanguageINIKey Property, page 290, **LanguageRegistryKey Property**, page 291, and **LoadBinaryResource Method**, page 292.

LocalizeForm Method

This method localizes a form by iterating through all resources in a resource group and loading each resource into a corresponding control property.

Syntax

object.LocalizeForm *Form*, *ResourceFile*, *ResourceGroup*

Remarks

This method works with both text and graphic properties. For example:

```
Form.Caption="Demo Application"
mnuFile.Caption="&File"
mnuFileOpen.Caption="&Open..."
imgApplicationBitmap.Picture=FILE:App.ico
...
```

Using this method, one line of code in your form localizes all the visual GUI controls on your form. To use this method, call it from the Load event for your form.

Example

The following example uses a Resource file called Forms that contains resource groups with the same names as the forms in the Me.Name application:

```
Private Sub Form_Load ()
    Res.LocalizeForm Me, "Forms", Me.Name
End Sub
```

See Also

LanguageINIKey Property, page 290, and **LanguageRegistryKey Property**, page 291.

Message Method

This method returns the resource for a specified resource ID. You do not specify the Resource file and resource group values within this method; to specify these values, call the SetDefaultMessageGroup method.

Syntax

```
result = object.Message(ResourceID, DefaultMessage, Substitutions...)
```

Part	Required	Data Type	Description
object	yes	Resource	Name of the Resource object.
ResourceID	yes	variant	ID for the resource.
DefaultMessage	yes	string	Message returned if the resource does not exist.
Substitutions	no	variant	One or more substitution parameters. Any "%" substitution place holders in the message are replaced by the specified parameters.

Example

A Resource file called "Messages" contains the following resource group:

```
[Misc]
RecordsFound="Records found: %1"
RangeError="The value must be in the range %1 to %2"
```

You can load and display these messages as follows:

```
Res.SetDefaultMessageGroup "Messages", "Misc"
MsgBox Res.Message("RecordsFound", "Records found: %1", icount)
MsgBox Res.Message("RangeError", "The value must be in the range _
                                  %1 to %2", ilow, ihigh)
```

Remarks

Before using this method, set the default Resource file and resource group by calling the SetDefaultMessageGroup method. After these values are set, you can call the Message method repeatedly without specifying the Resource file and resource group names each time.

The Substitutions argument is optional. Use it to pass any substitution parameters required by the message. If you do not pass enough substitution parameters to the message, the remaining parameters are replaced by "***".

Data Type

String

See Also

MessageEx Method, page 295, and **SetDefaultMessageGroup Method**, page 297.

MessageEx Method

This method returns the resource for a specified Resource file, group, and ID.

Syntax

```
result = object.MessageEx(ResourceFile, ResourceGroup, _
           ResourceID, DefaultMessage, Substitutions...)
```

Part	Required	Data Type	Description
object	yes	Resource	Name of the Resource object.
ResourceFile	yes	variant	Name of the Resource file.
ResourceGroup	yes	variant	Name of the resource group.
ResourceID	yes	variant	ID for the resource.
DefaultMessage	yes	string	Message returned if the resource does not exist.
Substitutions	no	variant	One or more substitution parameters. Any "%" substitution placeholders in the message are replaced by the specified parameters.

Example

A Resource file called "Messages" contains the following resource group:

```
[Misc]
RecordsFound="Records found: %1"
RangeError="The value must be in the range %1 to %2"
```

Load and display these messages as follows:

```
MsgBox Res.MessageEx("Messages", "Misc", "RecordsFound", _
                                 "Records found: %1", icount)
MsgBox Res.MessageEx("Messages", "Misc", "RangeError", _
    "The value must be in the range %1 to %2", ilow, ihigh)
```

Remarks

The Substitutions argument is optional. Use it to pass any substitution parameters required by the message. If you do not pass enough substitution parameters to the message, the remaining parameters are replaced by "***".

Data Type

String

See Also

Message Method, page 294.

ResourceFilePath Property

This property sets the file path used to search for Resource files; it is read or write.

Syntax

```
object.ResourceFilePath = value
```

or

```
result = object.ResourceFilePath
```

Example

```
.ResourceFilePath = "\\SERVER\Resources" & ";" & _
                    "C:\Program Files\Demos\Demo1"
```

To separate path names, use a semicolon (as shown above).

Remarks

Setting this property allows Resource files to reside in multiple locations:

- Store Resource files used by many applications on a shared network resource
- Store application-specific Resource files in the application's directory

SetDefaultMessageGroup Method

This method sets the default Resource file and resource group used by the Message method when loading resources.

Syntax

```
result = object.MessageEx(ResourceFile, ResourceGroup, _
           ResourceID, DefaultMessage, Substitutions...)
```

Part	Required	Data Type	Description
object	yes	Resource	Name of the Resource object.
ResourceFile	yes	variant	Name of the Resource file.
ResourceGroup	yes	variant	Name of the resource group.
ResourceID	yes	variant	ID for the resource.
DefaultMessage	yes	string	Message returned if the resource does not exist.
Substitutions	no	variant	One or more substitution parameters. Any “%” placeholders are replaced by the specified parameters.

Example

```
Res.SetDefaultMessageGroup RF_APP, "CustomMessages"
```

See Also

Message Method, page 294.

ResourceGroup Class

This class returns a list of resources in a resource group. It has the following properties:

Count Property

This property returns the number of resources in a resource group; it is read-only.

Syntax

```
result = object.Count
```

Remarks

If the specified resource group does not contain any resources, this property returns 0.

Example

```
Dim rg As ResourceGroup

Set rg = Res.GetResourceGroup("Forms", "frmOpen")
If rg Is Nothing Then
    Print "The resource group does not exist."
Else
    Print "The resource group contains" & rg.Count & " resources."
End If
```

ResourceFile Property

This property returns the name of a Resource file containing a resource group; it is read-only.

Syntax

```
result = object.ResourceFile
```

Remarks

The Resource file name is the name passed as a parameter to the GetResourceGroup method that created the ResourceGroup object.

Example

```
Dim rg As ResourceGroup

Set rg = Res.GetResourceGroup("Forms", "frmOpen")
If rg Is Nothing Then
    Print "The resource group does not exist."
Else
    Print rg.ResourceFile      ' Prints "Forms".
End If
```

ResourceGroup Property

This property returns the ID for a resource group; it is read-only.

Syntax

```
result = object.ResourceGroup
```

Remarks

The resource group ID is the ID passed as a parameter to the GetResourceGroup method that created the ResourceGroup object.

Example

```
Dim rg As ResourceGroup

Set rg = Res.GetResourceGroup("Forms", "frmOpen")
If rg Is Nothing Then
    Print "The resource group does not exist."
Else
    Print rg.ResourceGroup      ' Prints "frmOpen".
End If
```

ResourceID Property

This array property returns the resource IDs for resources in a resource group.

Syntax

```
result = object.ResourceID(index)
```

Remarks

The specified index value must be a value between 1 and the value of the Count property.

Example

```
Dim rg As ResourceGroup
Dim lindex As Long

Set rg = Res.GetResourceGroup("Forms", "frmOpen")
If rg Is Nothing Then
    Print "The resource group does not exist."
Else
    For lindex = 1 To rg.Count
        Print rg.ResourceI
```

See Also

Count Property, page 298.

SPECTRUM DISPATCH CLIENT CLASSES

This chapter describes the Spectrum Dispatch Client (SDC) and its associated classes. It contains information about each of the classes exposed in the SDC, as well as their methods and properties.

The following topics are covered:

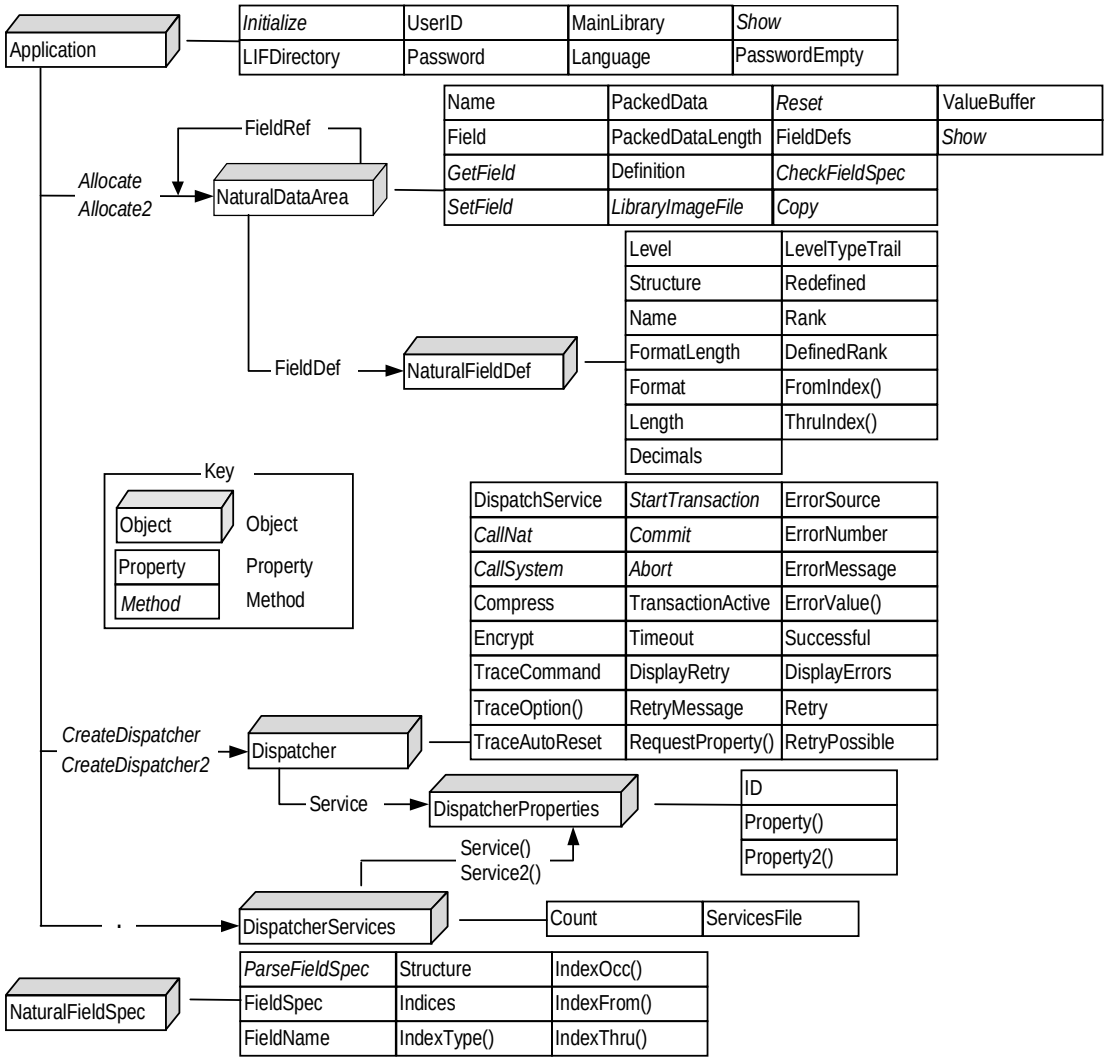
For additional information, see:

- **Creating Applications Without the Framework**, *Construct Spectrum SDK Reference*

Overview

The Spectrum Dispatch Client (SDC) provides the client/server data exchange that enables calls from a client to Natural subprograms running on a server. For more information about the SDC, **Using the Spectrum Dispatch Client**, *Construct Spectrum SDK Reference*.

The following diagram shows the structure of the SDC:



Structure of the Spectrum Dispatch Client

Application Class

The Application class is the highest level object for the Spectrum Dispatch Client (SDC). The client framework declares a global instance of the Application class in the SDCSupport.bas module.

The methods and properties of the Application class can be grouped as follows:

- **Dispatcher Creation**, page 303
- **Natural Data Area Allocation**, page 305
- **SDC Initialization**, page 306
- **User Identification**, page 308

The following sections describe the methods and properties within each group.

Dispatcher Creation

CreateDispatcher Method, page 303	**DispatcherServices Property**, page 304

CreateDispatcher Method

This method creates a Dispatcher object used to perform remote CallNats.

Syntax

```
Set result = object.CreateDispatcher(DispatchService)
```

Part	Required	Data Type	Description
object	yes	Application	Name of Application object.
Dispatch-Service	no	String	Name of a dispatch service definition in the SDC.ini file.
result		Dispatcher	Name of Dispatcher object.

This method creates a Dispatcher object that uses the dispatch service you specify. If the Dispatch Service parameter does not identify a valid dispatch service definition, this method returns Nothing.

Remarks

If you omit the DispatchService parameter, this method creates a Dispatcher object that uses the first dispatch service definition in the SDC.ini file. You can use the Dispatch-Services property to select a different dispatch service definition in the SDC.ini file.

See Also

DispatcherServices Property, page 304.

DispatcherServices Property

This property gets the collection of service definitions defined in the SDC.ini file; it is read-only.

Syntax

```
Set result = object.DispatcherServices ()
```

Part	Required	Data Type	Description
object	yes	Application	Name of Application object.
result		DispatcherServices	Name of DispatcherServices object to receive the collection of service definitions.

You must call the Refresh method to read changes.

Remarks

Changes to the SDC.ini file are not immediately reflected in the DispatcherServices object. For example:

- `Nat.DispatcherServices.Count` contains two service definitions.
- Add a new dispatch service definition using the Spectrum Service Manager.
- Nat.DispatcherServices.Count still contains two dispatch service definitions.
- Call Nat.DispatcherServices.Refresh.
- `Nat.DispatcherServices.Count` now contains three dispatch service definitions.

See Also

CreateDispatcher Method, page 303.

Natural Data Area Allocation

Allocate Method, page 305	**Show Method**, page 306

Allocate Method

This method creates a NaturalDataArea object that simulates a Natural data area.

Syntax

```
Set result = object.Allocate(DataArea, VSubstitutions…)
```

Part	Required	Data Type	Description
object	yes	Application	Name of Application object.
DataArea	yes	string	Name of data area to allocate. The data area must be defined in the main library image file or one of the library image files in the steplib chain.
DataAreaName	yes	string	Name of data area to allocate. This must be a multi-line string. End each line with a carriage-return/line-feed pair. This is optional on the last line.
VSubstitutions	no	variant array	V substitution list.
result		NaturalDataArea	Name of NaturalDataArea object.

Remarks

- Before you make any calls to this method, call the Initialize method to set the names of the LIF directory and main library.
- If the data area definition does not exist in the main library image file or one of the library image files in the steplib chain, a trappable runtime error occurs.
- All fields in the allocated data area are reset to their initial values.

See Also

Initialize Method, page 306, and **NaturalDataArea Class**, page 337.

Show Method

Displays a pop-up dialog showing the values of all fields in one or more data areas. The values can be edited.

Syntax

object.Show *DataAreas…*

Part	Required	Data Type	Description
object	yes	Application	Name of Application object.
DataAreas	yes	NaturalDataArea	One or more data areas to display.

Remarks

The Show method does not return until you close the dialog.

Example

```
Nat.Show custmsa, custmsr, cdaobj, cdpda_m
```

SDC Initialization

Initialize Method, page 306	**MainLibrary Property**, page 308
LIFDirectory Property, page 307	

Initialize Method

This method initializes the Spectrum Dispatch Client (SDC) by setting the location of the library image file directory and the name of the main library.

Syntax

object.Initialize *LIFDirectory*, *MainLibrary*

Part	Required	Data Type	Description
object	yes	Application	Name of Application object.
LIFDirectory	yes	string	Name of library image file directory.
MainLibrary	yes	string	Name of main library.

Remarks

Setting the library image file directory allows the SDC to find the application's library image files to load data area and application service definitions. If either the LIFDirectory or MainLibrary parameter is an empty string, an "Invalid procedure call" error occurs (error 5).

Do not use the .LIF extension in the MainLibrary parameter. For example, write code as:

```
Nat.Initialize "C:\DataFiles","LIBNAME"
```

not:

```
Nat.Initialize "C:\DataFiles", "LIBNAME.lif"
```

The name of the main library image file is derived from the LIF directory name and the main library name as follows:

```
LIFDirectory\MainLibrary.lif
```

For example, if you perform the following call:

```
Nat.Initialize "C:\Dev\Projects\Demo", "SYSNBSDE"
```

The name of the main library image file is:

```
C:\Dev\Projects\Demo\SYSNBSDE.lif
```

LIFDirectory Property

This property gets the name of the library image file directory; it is read-only. To set this property, use the Initialize method.

Syntax

```
result = object.LIFDirectory
```

Part	Required	Data Type	Description
object	yes	Application	Name of Application object.
result		string	Name of library image file directory.

Remarks

If the library image file directory has not been set, this property returns an empty string.

MainLibrary Property

This property gets the name of the main library; it is read-only. To set this property, use the Initialize method.

Syntax

```
result = object.MainLibrary
```

Part	Required	Data Type	Description
object	yes	Application	Name of Application object.
result		string	Name of main library.

Remarks

If the main library has not been set, this property will return an empty string.

User Identification

Language Property

This property gets or sets the language code; it is read or write. The language code indicates the language code used in calls to the server (*Language value, such as 1 for English, 2 for German).

The Spectrum dispatch service uses the language code to return localized message text to the client. The Spectrum dispatch service also sets *Language to this value before calling the application service (subprogram), allowing the service to return localized message text.

Syntax

```
object.Language = value
```

or

```
result = object.Language
```

Part	Required	Data Type	Description
object	yes	Application	Name of Application object.
value	yes	Integer	Value of *Language (as defined in Natural) to set.
result		Integer	Current value of *Language.

Remarks

If the Language property contains 0 (the default value, if it has not been assigned), the Spectrum dispatch service defaults to English (unless a language is defined in the Spectrum Administration subsystem for that dispatch service).

For more information about the default language code for dispatch services, refer to *Construct Spectrum SDK Reference*.

See Also

Password Property, page 309, and **UserID Property**, page 311.

Password Property

This property sets the password for calls to the server; it is write-only.

Syntax

```
object.Password = value
```

Part	Required	Data Type	Description
object	yes	Application	Name of Application object.
value	yes	string	Password to set.

Remarks

The Spectrum dispatch service uses this property to authenticate the user performing the request.

See Also

UserID Property, page 311, and **PasswordEmpty Property**, page 310.

PasswordEmpty Property

This method returns information about the Password property setting. It indicates whether password protection is being used.

Syntax

```
result = object.PasswordEmpty
```

Part	Required	Data Type	Description
result	yes	Boolean	True if the Password property is empty, False if not.
object	yes	Application	Name of Application object.

Remarks

Because the Password property can be assigned but not read, the PasswordEmpty property returns whether or not the Password property has been set.

Example

```
Nat.Password = "PASSWORD"
Debug.Print Nat.PasswordEmpty       ' Prints False.

Nat.Password = ""
Debug.Print Nat.PasswordEmpty       ' Prints True.
```

See Also

Password Property, page 309.

UserID Property

This property gets or sets the user ID for calls to the server; it is read or write.

Syntax

object.UserID = *value*

or

result = *object*.UserID

Part	Required	Data Type	Description
object	yes	Application	Name of Application object.
value	yes	string	User ID to set.
result		string	Current value of user ID.

Remarks

You must set this property to a valid user ID before calling the server. If the user ID has an associated password, you must also set the Password property; otherwise, the Spectrum dispatch service rejects the request.

See Also

Password Property, page 309.

Dispatcher Class

This class handles communication between the client and the server; it provides methods and properties to interact with the Spectrum Dispatch Service.

The methods and properties for the Dispatcher class can be grouped as follows:

- **Compression and Encryption**, page 312
- **Database Transaction Management**, page 314
- **Error Information**, page 316
- **Remote Subprogram Invocation**, page 321
- **Response Timeout and Retry Handling**, page 326
- **Tracing**, page 330

The following sections describe the methods and properties within each group.

Compression and Encryption

Compress Property, page 312	**Encrypt Property**, page 313

Compress Property

This property gets or sets the Compress flag for request data sent to the Spectrum Dispatch Service; it is read or write.

If True, this property compresses the request data; compressing data can reduce transmission time, especially over slow network connections.

Syntax

object.Compress = *value*

or

result = *object*.Compress

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
value	yes	Boolean	If True, request data is compressed; if False, data is not compressed.
result		Boolean	Current value of Compress flag (True or False).

Remarks

The compression algorithm reduces sequences of repeating characters, which is common in Natural parameter data areas.

This property affects only the request data sent to the Spectrum Dispatch Service. To compress the response data, generate the subprogram proxy and mark the Compress flag on the model specification panel.

Encrypt Property

This property gets or sets the Encrypt flag for request data sent to the Spectrum Dispatch Service; it is read or write.

Syntax

object.Encrypt = *value*

or

result = *object*.Encrypt

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
value	yes	Boolean	If True, request data is encrypted; if False, data is not encrypted.
result		Boolean	Current value of Encrypt flag (True or False).

Remarks

This property only affects the request data sent to the Spectrum Dispatch Service. To encrypt the response data, generate the subprogram proxy and mark the Encrypt flag on the model specification panel.

Database Transaction Management

Abort Method

This method aborts any pending database updates by sending a BACKOUT TRANSACTION request to the Spectrum Dispatch Service; this releases the current service and the client application no longer has exclusive access to it.

Syntax

object.Abort

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.

See Also

Commit Method, page 314, **StartTransaction Method**, page 315, and **TransactionActive Property**, page 315.

Commit Method

This method commits any pending database updates by sending an END TRANSACTION request to the Spectrum Dispatch Service; this releases the current service and the client application no longer has exclusive access to it.

Syntax

object.Commit

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.

See Also

Abort Method, page 314, **StartTransaction Method**, page 315, and **TransactionActive Property**, page 315.

StartTransaction Method

This method starts a transaction on the server. To end the transaction, use the Abort or Commit methods.

Syntax

```
object.StartTransaction
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.

Remarks

All calls within a transaction to the CallNat and CallSystem methods are sent to the same dispatch server, which is dedicated to this client.

See Also

Abort Method, page 314, **Commit Method**, page 314, and **TransactionActive Property**, page 315.

TransactionActive Property

This property returns information about whether a dispatch service is currently active; it is read-only. If a transaction is active, this property returns True; if a transaction is not active, it returns False.

Syntax

```
result = object.TransactionActive
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
result		Boolean	Current value of TransactionActive flag (True or False).

See Also

Abort Method, page 314, **Commit Method**, page 314, and **TransactionActive Property**, page 315.

Error Information

DisplayErrors Property

This property gets or sets the DisplayErrors flag; it is read or write.

The DisplayErrors flag indicates whether communication errors are displayed within a pop-up message box from inside the CallNat or CallSystem method.

Syntax

object.DisplayErrors = *value*

or

result = *object*.DisplayErrors

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
value	yes	Boolean	If True, errors are displayed; if False, errors are not displayed.
result		Boolean	Current value of DisplayErrors flag (True or False).

Remarks

Error information is always returned in the error properties: ErrorMessage, ErrorNumber, ErrorSource, ErrorValue, and Successful. However, when DisplayErrors is set to True, and a communication error occurs inside the CallNat or CallSystem methods, the SDC also displays the error message in a message box.

See Also

- **CallNat Method**, page 321
- **CallSystem Method**, page 322
- **ErrorMessage Property**, page 317
- **ErrorNumber Property**, page 318
- **ErrorSource Property**, page 318
- **ErrorValue Property**, page 319
- **Successful Property**, page 320

ErrorMessage Property

This property gets the message text associated with the last communications error; it is read-only.

Syntax

```
result = object.ErrorMessage
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
result		string	Error message text associated with the last communications error.

See Also

- **ErrorNumber Property**, page 318
- **ErrorSource Property**, page 318
- **ErrorValue Property**, page 319
- **Successful Property**, page 320

ErrorNumber Property

This property gets the message number associated with the last communications error; it is read-only.

Syntax

```
result = object.ErrorNumber
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
result		string	Message number associated with the last communications error.

Remarks

The format of the message number depends on the value of the ErrorSource property. For more information on communications errors, see **Debugging Your Client/Server Application**, *Construct Spectrum SDK Reference.*

See Also

- **ErrorMessage Property**, page 317
- **ErrorSource Property**, page 318
- **ErrorValue Property**, page 319
- **Successful Property**, page 320

ErrorSource Property

This property gets the source (where the error originated) associated with the last communications error; it is read-only.

Syntax

```
result = object.ErrorSource
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
result		string	Source associated with the last communications error.

Possible ErrorSource values are:

Value	Error occurred in the:
ETB	Broker stub on the client.
NAT	Subprogram or subprogram proxy on the server.
SPE	Client or server components of Spectrum.

See Also

- **ErrorMessage Property**, page 317
- **ErrorSource Property**, page 318
- **ErrorValue Property**, page 319
- **Successful Property**, page 320

ErrorValue Property

This property gets the substitution parameters associated with the last communications error; it is read-only.

Syntax

```
result = object.ErrorValue(index)
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
index	yes	Integer	Ordinal number for the substitution parameter. The first substitution is 1.
result		string	Substitution values associated with last communications error.

Remarks

The substitution values are substituted into the error message returned by the ErrorMessage property. To supply your own error message, access the substitution parameters yourself.

See Also

- **ErrorMessage Property**, page 317
- **ErrorSource Property**, page 318
- **ErrorValue Property**, page 319
- **Successful Property**, page 320

Successful Property

This property gets information about whether the last CallNat or CallSystem method was successful; it is read-only. If the method was successful, this property returns True; if the method was not successful, it returns False.

Syntax

```
result = object.Successful
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
result		Boolean	Current value of Successful flag (True or False).

Remarks

The value returned by this property is the same as the value returned from the CallNat or CallSystem methods.

See Also

- **CallNat Method**, page 321
- **CallSystem Method**, page 322
- **ErrorMessage Property**, page 317
- **ErrorSource Property**, page 318
- **ErrorValue Property**, page 319

Remote Subprogram Invocation

CallNat Method

This method performs a remote CallNat to invoke a Natural subprogram on the server.

Syntax

```
result = object.CallNat(AppServiceName, DataAreas…)
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
AppService	yes	string	Application service to call.
AppServiceName	yes	string	Name of Application service to call.
DataAreas	no	variant	Name of one or more NaturalDataArea objects.
result		Boolean	If successful, returns True; if not successful, returns False.

Remarks

The AppServiceName value must correspond to an application service definition in the main library image file or one of the library image files in its steplib chain. It can include an optional method name using the format:

```
AppServiceName.MethodName
```

For example:

```
disp.CallNat "CUSTOMER.GET", custpda, _
                             custpdaid, _
                             custpdr, _
                             cdaobj, _
                             cdpdam
```

If you use the method name, you must define the method in the application service definition. If you do not use the method name, a method called "DEFAULT" is used. Ensure that this default method is defined in the application service definition.

Each method definition specifies the number of level 1 structures expected by the subprogram that implements the method.

The application service must consist of the domain, object, version, method, block count, and block identifiers. You can create this from an application service definition in a LIF file. For example:

```
[AppService CUSTOMER]
Domain=DEMO
Object=CUSTOMER
Version=1.1.1
Method=BROWSE,,5,1+3+4
Method=GET,,5,2+4
Method=NEXT,,5,2+4
```

becomes:

```
Private Const APPSERVICE_CUSTOMER = "DEMO,CUSTOMER,1.1.1,"
Public Const APPSERVICE_CUSTOMER_BROWSE = APPSERVICE_CUSTOMER & _
                                          "BROWSE,5,1+3+4"
Public Const APPSERVICE_CUSTOMER_GET = APPSERVICE_CUSTOMER & _
                                       "GET,5,2+4"
Public Const APPSERVICE_CUSTOMER_NEXT = APPSERVICE_CUSTOMER & _
                                        "NEXT,5,2+4"
```

Notice that in the LIF file, the method definition contains two method names (the second can be optional): the first method name is used in the CallNat statement, and the second method name corresponds to the one defined on the application service definition record in Spectrum.

CallSystem Method

This method either:

- Sends system commands directly to the Spectrum Dispatch Service

  or

- Invokes an arbitrary subprogram proxy by specifying its domain, object, version, and method

Syntax

```
result = object.CallSystem(DomainName, ObjectName, Version, _
MethodName, SendData, ReceiveData)
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
DomainName	yes	string	Name of domain.
ObjectName	yes	string	Name of object.

Part	Required	Data Type	Description (continued)
Version	yes	long	Version coded into a long Integer.
MethodName	yes	string	Name of method.
SendData	yes	string	Data to send to the dispatch service or subprogram proxy.
ReceiveData	yes	string	Data received from the dispatch service or subprogram proxy.
result		Boolean	If successful, returns True; if not successful, returns False.

Remarks

The product identifier consists of three numbers that are separated by periods, for example, 1.2.4. These numbers represent major, minor and release information for the product. Code this value in a long integer with the form:

```
mmnnrr
```

where:

`mm` Is the major number.

`nn` Is the minor number.

`rr` Is the release number.

Example

```
CallSystem "DEMO", "CUSTOMER”, 10204,"GET", senddata,receivedata
```

The leading zero in the major number can be omitted (see “10204” in example).

See Also

CallNat Method, page 321.

DispatchService Property

This property gets or sets the ID for the dispatch service a dispatcher object is communicating with; it is read or write.

Syntax

```
object.DispatchService = value
```

or

```
result = object.DispatchService
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
value	yes	string	Dispatch service ID to set.
result		string	Current dispatch service ID.

Remarks

The value assigned to this property must correspond to a dispatch service definition ID defined in the Spectrum Service Manager program.

See Also

Service Property, page 325.

RequestProperty Property

This property returns information gathered during the last CallNat or CallSystem request sent to the dispatch service.

Syntax

```
result = object.RequestProperty(index)
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
index	yes	string	Name of property. For a list of possible values, see **RequestProperty Property**, *Construct Spectrum SDK Reference*.
result		string	Property value for the last CallNat or CallSystem method.

Remarks

The SDC saves detailed information about each request sent to the dispatch service. To obtain information about the request, use this array property after performing the CallNat or CallSystem method.

See Also

CallNat Method, page 321, and **CallSystem Method**, page 322.

Service Property

This property gets a reference to the dispatch service that a dispatcher object is communicating with; it is read-only.

Syntax

```
Set result = object.Service
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
result		DispatcherProperties	Reference to dispatch service with which the Dispatcher object is communicating.

See Also

DispatchService Property, page 324.

Response Timeout and Retry Handling

DisplayRetry Property

This property gets or sets whether the SDC displays a retry message if the dispatch service does not send a response within the timeout period; it is read or write.

Syntax

```
object.DisplayRetry = value
```

or

```
result = object.DisplayRetry
```

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
value	yes	Boolean	If True, retry message is displayed; if False, message is not displayed.
result		Boolean	Current value of DisplayRetry flag (True or False).

Remarks

If the dispatch service does not send a response to the client within the timeout period (set by the Timeout property), the CallNat or CallSystem method either returns to the caller or displays a retry message — depending on the DisplayRetry value.

Use the RetryMessage property to set the message.

See Also

- **CallNat Method**, page 321
- **CallSystem Method**, page 322
- **RetryMessage Property**, page 328
- **Timeout Property**, page 329

Retry Property

This property gets or sets whether to retry the last CallNat or CallSystem method; it is read or write. If True, the last CallNat or CallSystem method is resumed (if the method returned with an error that is retryable).

Syntax

object.Retry = *value*

or

result = *object*.Retry

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
value	yes	Boolean	If True, the last CallNat or CallSystem method is retried; if False, the method is not retried.
result		Boolean	Current value of Retry flag (True or False).

Remarks

After performing the CallNat or CallSystem method, this property can only be set to True if the RetryPossible property returns True. Otherwise, a trappable runtime error occurs.

See Also

CallNat Method, page 321, **CallSystem Method**, page 322, and **RetryPossible Property**, page 328.

RetryMessage Property

This property gets or sets the message displayed when the server is not responding and the DisplayRetry property is True. Word this message so that a "Yes" response continues waiting and a "No" response stops waiting and returns control to the caller.

Syntax

object.RetryMessage = *value*

or

result = *object*.RetryMessage

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
value	yes	string	Message to display to the user.
result		string	Current message displayed to the user.

See Also

DisplayRetry Property, page 326.

RetryPossible Property

This property gets information about whether the last CallNat or CallSystem method can be retried; it is read-only.

Syntax

result = *object*.RetryPossible

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
result		Boolean	If True, the last CallNat or CallSystem method can be resumed; if False, the last method cannot be retried.

See Also

CallNat Method, page 321, **CallSystem Method**, page 322, and **RetryPossible Property**, page 328.

Timeout Property

This property gets or sets the response timeout value in seconds; it is read or write.

Syntax

object.Timeout = *value*

or

result = *object*.Timeout

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
value	yes	Integer	Response timeout value (either set programmatically by assigning a value from 0 to 32767 to this property, or supplied by the Spectrum Service Manager by assigning a negative number).
result		Integer	Current response timeout value.

Remarks

If the result is 0 or a positive value, the Timeout value has been overridden programmatically; if the result is a negative value, this property returns the negative of the value set in the Spectrum Service Manager.

See Also

CallNat Method, page 321, **CallSystem Method**, page 322, and **RetryPossible Property**, page 328.

Tracing

TraceAutoReset Property

This property gets or sets whether the TraceCommand and TraceOption flags are automatically reset if the next CallNat or CallSystem method returns successfully; it is read or write.

Syntax

object.TraceAutoReset = *value*

or

result = *object*.TraceAutoReset

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
value	yes	Boolean	If True, the TraceCommand and TraceOptions flags are automatically reset if the next CallNat or CallSystem method returns successfully; if False, these flags are not reset.
result		Boolean	Current value of TraceAutoReset flag (True or False).

See Also

- **CallNat Method**, page 321
- **CallSystem Method**, page 322
- **TraceCommand Property**, page 331
- **TraceOption Property**, page 331

TraceCommand Property

This property gets or sets the Trace command sent to the server; it is read or write.

Syntax

object.TraceCommand = *value*

or

result = *object*.TraceCommand

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
value	yes	string	Trace command sent to the server.
result		string	Current value of Trace command sent to server.

See Also

- **CallNat Method**, page 321
- **CallSystem Method**, page 322
- **TraceAutoReset Property**, page 330
- **TraceOption Property**, page 331

TraceOption Property

This property gets or sets the Trace options sent to the server; it is read or write.

Syntax

object.TraceOption (*index*) = value

or

result = *object*.TraceOption (*index*)

Part	Required	Data Type	Description
object	yes	Dispatcher	Name of Dispatcher object.
index		Integer	Index for Trace options sent to the server.
value	yes	Integer	Value of Trace options sent to the server.
result		Integer	Current value for Trace options sent to server.

See Also

- **CallNat Method**, page 321
- **CallSystem Method**, page 322
- **TraceAutoReset Property**, page 330
- **TraceCommand Property**, page 331

DispatcherProperties Class

This class defines objects that return information about the configuration of a Spectrum Service. The DispatcherProperties class has the following properties:

ID Property, page 332	**Refresh Method**, page 334
Property Property, page 333	

The following sections describe these properties.

ID Property

This property gets the ID for the service definition; it is read-only.

Syntax

```
result = object.ID
```

Part	Required	Data Type	Description
object	yes	DispatcherProperties	Name of DispatcherProperties object.
result		string	Dispatch service definition ID.

See Also

Property Property, page 333.

Property Property

This property gets the configuration setting value in a service definition; it is read-only.

Example

```
Dim dp As DispatcherProperties

Set dp = Nat.DispatcherServices(1)
Debug.Print dp.ID
Debug.Print dp("Name")
Debug.Print dp.Property("BrokerID")
```

Syntax

```
result = object.Property(Name)
```

Part	Required	Data Type	Description
object	yes	DispatcherProperties	Name of DispatcherProperties object.
name	yes	string	Name of a property to read. For a list of valid values, see **Remarks**.
result		string	Configuration setting in the service definition.

Remarks

Name can be any of the following values:

Value	Description
Description	Description of dispatch service.
BrokerID	Broker ID used by the dispatch service.
ServerClass	Name of server class.
ServerName	Name of server.
Service	Name of service.
Timeout	Timeout value.
PreferredCharSet	ASCII or EBCDIC option (character set in which the dispatch service prefers to receive data).
SecurityMode	Security mode (SECURED, UNSECURED, or BROKER).

Value	Description (continued)
Type	Type of service (DIS411, DIS421, ATT421, or CON421).
result	Configuration setting in the dispatch service definition.

See Also

ID Property, page 332.

Refresh Method

This method loads the current service definition values in the SDC.ini file into the DispatcherProperties object.

Syntax

object.Refresh

Part	Required	Data Type	Description
object	yes	DispatcherProperties	Name of DispatcherProperties object.

Example

```
Dim dp As DispatcherProperties

Set dp = Nat.DispatcherServices(1)
dp.Refresh
```

See Also

Service Property, page 335.

DispatcherServices Class

This class is a collection of all service definitions defined to the Spectrum service manager.

The DispatcherServices class has the following properties:

Count Property, page 335	**ServicesFile Property**, page 336
Service Property, page 335	

The following sections describe these properties.

Count Property

This property gets the number of service definitions in the SDC.ini file; it is read-only.

Syntax

```
result = object.Count
```

Part	Required	Data Type	Description
object	yes	DispatcherServices	Name of DispatcherServices object.
result		Integer	Number of service definitions in the SDC.ini file.

See Also

Service Property, page 335, and **ServicesFile Property**, page 336.

Service Property

This property gets a DispatcherProperties object containing properties for a specified service definition; it is read-only.

Example

```
Dim dp As DispatcherProperties

Set dp = Nat.DispatcherServices(1)
Set dp = Nat.DispatcherServices("DISPATCHER")
```

Syntax

```
Set result = object.Service(ServiceID)
```

Part	Required	Data Type	Description
object	yes	DispatcherServices	Name of DispatcherServices object.
result		DispatcherProperties	Name of DispatcherProperties object containing properties for the specified dispatch service.
ServiceID	yes	variant	Service definition ID.

Remarks

`ServiceID` can be one of the following:

- A numeric value from 1 to the value of the Count property
- A string identifying a dispatch service definition ID

See Also

Count Property, page 335, and **ServicesFile Property**, page 336.

ServicesFile Property

This property gets the full file name of the SDC.ini file; it is read-only.

Syntax

```
result = object.ServicesFile
```

Part	Required	Data Type	Description
object	yes	DispatcherServices	Name of DispatcherServices object.
result		string	Full file name of SDC.ini file.

See Also

Count Property, page 335, and **ServicesFile Property**, page 336.

NaturalDataArea Class

This class defines the properties and methods for the simulated Natural data areas. Each instance of this class stores details about its structure and maintains field values for a Natural data area. A client application can create as many instances of the same or different data areas as required.

The methods and properties used by the NaturalDataArea class can be grouped as follows:

- **Informational**, page 337
- **Field Access**, page 342

The following sections describe the methods and properties within each group.

Informational

Definition Property, page 337	**LibraryImageFile Property**, page 339
FieldDef Property, page 338	**Name Property**, page 340
FieldDefs Property, page 339	**ValueBuffer Method**, page 341

Definition Property

This property gets a multiple-line string that contains the entire data area definition as read from the library image file; it is read-only.

Syntax

```
result = object.Definition
```

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
result		string	Data area definition as read from the library image file.

See Also

- **FieldDef Property**, page 338
- **FieldDefs Property**, page 339
- **LibraryImageFile Property**, page 339
- **Name Property**, page 340
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361

FieldDef Property

This property gets a NaturalFieldDef object that defines a field. If the field is part of an array, any supplied index values are ignored; it is read-only.

Syntax

```
Set result = object.FieldDef (index)
```

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
result		NaturalFieldDef	Data area definition as read from the library image file.
index	yes	Integer or string	One of the following: • Numeric index to retrieve the field definition using its PDA line number • String index to retrieve the field definition using its PDA name

See Also

- **FieldDefs Property**, page 339
- **LibraryImageFile Property**, page 339
- **Name Property**, page 340
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361

FieldDefs Property

This property gets the number of field definitions in a data area definition; it is read-only.

Syntax

```
result = object.FieldDefs
```

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
result		Integer	Number of field definitions in the data area definition.

See Also

- **Definition Property**, page 337
- **FieldDef Property**, page 338
- **LibraryImageFile Property**, page 339
- **Name Property**, page 340
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361

LibraryImageFile Property

This property gets the full file name for the library image file from which a data area definition was loaded; it is read-only.

Syntax

```
result = object.LibraryImageFile
```

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
result		string	Full file name for the library image file from which the data area definition was loaded.

See Also

- **Definition Property**, page 337
- **FieldDef Property**, page 338
- **Name Property**, page 340
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361

Name Property

This property gets the name of the Natural data area represented by an object (the name that was passed to the Allocate method); it is read-only.

Syntax

```
result = object.Name
```

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
result		string	Name of Natural data area represented by the object.

See Also

- **Allocate Method**, page 305
- **Definition Property**, page 337
- **FieldDef Property**, page 338
- **LibraryImageFile Property**, page 339
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361

ValueBuffer Method

Sets or returns a copy of the internal block of memory that stores the field values (the value buffer).

Syntax

```
result = object.ValueBuffer
object.ValueBuffer = value
```

Part	Required	Data Type	Description
result		byte array	Copy of value buffer.
object	yes	NaturalDataArea	Name of NaturalDataArea object.
value		byte array	Value buffer to replace the internal value buffer.

Remarks

This property is useful when you want to copy the field value from one data area to another, as in the example. Both data areas must have the same size and compatible field formats.

Example

```
data1.ValueBuffer = data2.ValueBuffer
```

See Also

Copy Method, page 343.

Field Access

CheckFieldSpec Method

This method checks whether a field name is defined in a data area. If the field name is invalid, a runtime error occurs.

Syntax

object.CheckFieldSpec *fieldspec*

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
fieldspec	yes	NaturalFieldSpec	Name of NaturalFieldSpec object.

Example

```
Dim nfs As New NaturalFieldSpect
nfs.ParseFieldSpec"ROW(1)"
On Error Resume Next
dataarea1.CheckFieldSpec nfs
If Err.Number then
   MsgBox Err.Description
Else
   ...
End If
```

See Also

- **GetField Method**, page 346
- **Field Property**, page 344
- **FieldRef Property**, page 345
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361
- **PackedData Property**, page 347
- **PackedDataLength Property**, page 348
- **Reset Method**, page 349
- **SetField Method**, page 350

Copy Method

Creates an identical copy of a NaturalDataArea object, which then has a separate lifetime.

Syntax

object.Copy

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.

Remarks

This method creates a copy of a NaturalDataArea object with the same definition and field values as the original object. Field values changed in one do not affect the other. This differs from the FieldRef property. The FieldRef property also creates a Natural-DataArea object, but the new object shares field values with the original.

Example

```
Dim data1 As NaturalDataArea
Dim data2 As NaturalDataArea

' Allocate a data area.
Set data1 = Nat.Allocate2(DATAAREA_CSASTD)

' Create a copy of this data area.
Set data2 = data1.Copy()
```

See Also

FieldRef Property, page 345, and **SDC Initialization**, page 306.

Field Property

This property gets or sets the value in a field. It receives a field name as a parameter. If the field is part of an array, you must also specify index values as part of the field name; this property is read or write.

Syntax

```
object.Field (fieldname) = value
```

or

```
result = object.Field(fieldname)
```

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
value	yes	string	Value to set in the field.
result		variant	Current value in the field.
fieldname	yes	string	Name of field to read or write.

See Also

- **CheckFieldSpec Method**, page 342
- **GetField Method**, page 346
- **FieldRef Property**, page 345
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361
- **PackedData Property**, page 347
- **Reset Method**, page 349
- **Reset Method**, page 349
- **SetField Method**, page 350

FieldRef Property

This property creates a new NaturalDataArea object containing a subset of fields, which effectively creates two data areas that refer to the same data. It is read-only.

Syntax

```
Set object2 = object.FieldRef (fieldname)
```

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
object2	yes	NaturalDataArea	Name of new NaturalDataArea object.
fieldname	yes	string	Name of a field in the NaturalDataArea object.

See Also

- **CheckFieldSpec Method**, page 342
- **GetField Method**, page 346
- **Field Property**, page 344
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361
- **PackedData Property**, page 347
- **PackedDataLength Property**, page 348
- **Reset Method**, page 349
- **SetField Method**, page 350

GetField Method

This method reads the value in a field. It is similar to the Field property, except index values are specified as optional parameters (not part of the field name).

Syntax

```
result = object.GetField(fieldname, index1, index2, index3)
```

Part	Required	Data Type	Description
object	yes	string	Name of a field in the PDA.
index1	no	Integer	Value of index1, if the field is an array.
index2	no	Integer	Value of index2, if the field is an array.
index3	no	Integer	Value of index3, if the field is an array.
result	no	variant	Current value in the field.

Example

```
With dataarea1
  Print .GetField("CUSTOMER-NUMBER")
  Print .GetField("PHONE-NUMBER", 1)
  Print .GetField("STREET", 1, 1)
End With
```

See Also

- **CheckFieldSpec Method**, page 342
- **Field Property**, page 344
- **FieldRef Property**, page 345
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361
- **PackedData Property**, page 347
- **PackedDataLength Property**, page 348
- **Reset Method**, page 349
- **SetField Method**, page 350

PackedData Property

This property gets or sets field values for an entire data area as an alphanumeric string; it is read or write.

Syntax

object.PackedData = *value*

or

result = *object*.PackedData

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
value	yes	string	Field values to set for the data area.
result		string	Current field values for the data area.

Remarks

Assigning an alphanumeric string to this property replaces the field values in the specified data area with the values in the string. The assigned string must be the correct length.

For example, you can use this property to copy the field values from one data area to another if both have the same structure:

```
dataarea2.PackedData = dataarea1.PackedData.
```

See Also

- **CheckFieldSpec Method**, page 342
- **Field Property**, page 344
- **FieldRef Property**, page 345
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361
- **PackedDataLength Property**, page 348
- **Reset Method**, page 349
- **SetField Method**, page 350

PackedDataLength Property

This property gets the length of the packed data in a NaturalDataArea object. It is read-only.

Syntax

```
result= object.PackedDataLength
```

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
result		long	Length of packed data.

Example

```
If Len(pdata) <> dataarea1.PackedDataLength Then
    MsgBox "The packed data is not the right length."
Else
    dataarea1.PackedData = pdata
End If
```

See Also

- **CheckFieldSpec Method**, page 342
- **Field Property**, page 344
- **FieldRef Property**, page 345
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361
- **PackedData Property**, page 347
- **Reset Method**, page 349
- **SetField Method**, page 350

Reset Method

This method resets the fields in a data area to their default values.

Syntax

object.Reset

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.

Remarks

You can pass a field name to the Reset method to reset only that field. For example:

```
dataarea1.Reset "CUSTOMER-NUMBER"
```

You can also reset structures and multiple occurrences of an array. For example:

```
dataarea1.Reset "CUSTOMER"
dataarea1.Reset "STREET(*,*)"
dataarea1.Reset "STREET(1,*)"
```

See Also

- **CheckFieldSpec Method**, page 342
- **GetField Method**, page 346
- **Field Property**, page 344
- **FieldRef Property**, page 345
- **NaturalFieldDef Class**, page 352
- **NaturalFieldSpec Class**, page 361
- **PackedData Property**, page 347
- **PackedDataLength Property**, page 348
- **SDC Initialization**, page 306
- **SetField Method**, page 350

SetField Method

This method writes a field value in a NaturalDataArea object. It is similar to the Field property, except index values are optional (not part of the field name).

Syntax

```
object.SetField value, fieldname, index1, index2, index3)
```

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.
value	yes	variant	Value to set in the field.
fieldname	yes	string	Name of a field in the PDA.
index1	no	Integer	Value of index1, if the field is an array.
index2	no	Integer	Value of index2, if the field is an array.
index3	no	Integer	Value of index3, if the field is an array.

Example

```
 With dataarea1
.SetField(10001, "CUSTOMER-NUMBER")
.SetField("4165551234", "PHONE-NUMBER", 1)
.SetField("134 Hill Blvd.", "STREET", 1, 1)
 End With
```

See Also

- **CheckFieldSpec Method**, page 342
- **Field Property**, page 344
- **FieldRef Property**, page 345
- **GetField Method**, page 346
- **PackedData Property**, page 347
- **PackedDataLength Property**, page 348

Show Method

This method displays a pop-up dialog showing the values of all the fields in a data area. The values can be edited.

Syntax

object.Show

Part	Required	Data Type	Description
object	yes	NaturalDataArea	Name of NaturalDataArea object.

Remarks

The Show method does not return until you close the dialog.

Example

```
custmsa.Show
```

NaturalFieldDef Class

This class returns information about a single field in a data area definition. The FieldDef property for the NaturalDataArea class creates and returns an instance of this class.

The NaturalFieldDef class has the following properties, which are read-only:

Decimals Property, page 352	**LevelTypeTrail Property**, page 357
DefinedRank Property, page 353	**Name Property**, page 358
Format Property, page 353	**Rank Property**, page 358
FormatLength Property, page 354	**Redefined Property**, page 359
FromIndex Property, page 355	**Structure Property**, page 359
Length Property, page 356	**ThruIndex Property**, page 360
Level Property, page 356	

Decimals Property

This property gets the decimals portion of a Natural format and length string. It is read-only.

Syntax

`result = object.Decimals`

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
result		Integer	Number of decimal places in the Natural format and length string.

Remarks

If the format is not N (numeric) or P (packed numeric), this property returns 0.

See Also

Format Property, page 353, **FormatLength Property**, page 354, and **Length Property**, page 356.

DefinedRank Property

This property gets the number of dimensions defined for a field in a data area definition. This property is similar to the Rank property, except it returns the number of dimensions — regardless of any structure arrays the field may be part of.

Syntax

```
result = FieldDef("fieldname").DefinedRank
```

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
result		Integer	Number of dimensions defined for the field in the data area definition.

See Also

Rank Property, page 358.

Format Property

This property gets the format portion of a Natural format and length string (for example, A for alphanumeric, B for binary).

Syntax

```
result = object.Format
```

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
result		string	Natural format letter.

See Also

Decimals Property, page 352, **FormatLength Property**, page 354, and **Length Property**, page 356.

FormatLength Property

This property gets a Natural format and length string.

Syntax

```
result = object.FormatLength
```

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
result		string	Natural format and length string.

Example

Assume the following data area field definitions:

```
01 EMPLOYEE
  02 PID (P7)
  02 FIRST-NAME (A20)
  02 SALARY (P7.2)
  02 HIRE-DATE (D)
```

The following example code refers to these field definitions:

```
With employee
    Print .FieldDef("PID").FormatLength
    Print .FieldDef("FIRST-NAME").FormatLength
    Print .FieldDef("SALARY").FormatLength
    Print .FieldDef("HIRE-DATE").FormatLength
End With
    Prints: P7
            A20
            P7.2
            D
```

See Also

Decimals Property, page 352, and **Length Property**, page 356.

FromIndex Property

This property gets the low index value for each dimension of an array field. Use this property in conjunction with the ThruIndex property to determine the low and high index values.

Syntax

`result = object.FromIndex (index)`

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of FieldDef object.
index	yes	Integer	Dimension number (1, 2, or 3).
result		Integer	Low index value for the specified dimension of the array field.

Example

```
01 VALUES(N10/1:10,5:7)
With data.FieldDef("VALUES")
    For i = 1 To .Rank
        Print .FromIndex(i) & ":" & .ThruIndex(i)
    Next
End With
        Prints: 1:10
                5:7
```

See Also

DefinedRank Property, page 353, **Rank Property**, page 358, and **ThruIndex Property**, page 360.

Length Property

This property returns the length portion of the Natural format and length string.

Syntax

```
result = object.Length
```

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
result		Integer	Length portion of Natural format and length string.

See Also

Decimals Property, page 352, **Format Property**, page 353, and **FormatLength Property**, page 354.

Level Property

This property gets the level number for a field in a data area definition.

Syntax

```
result = object.Level
```

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
result		Integer	Level number for the field in the data area definition.

See Also

LevelTypeTrail Property, page 357.

LevelTypeTrail Property

This property gets a string you can use to determine the nesting of a field in a data area definition. The returned string contains one of the following characters for each level:

- F (field)
- S (structure)
- R (redefine)
- X (filler)

Syntax

```
result = object.LevelTypeTrail
```

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
result		Integer	Level number for the field in the data area definition.

Example

Assume the following data area field definitions:

```
01 ROW (1:10)
   02 ID (N6)
   02 ACCOUNT-NO (A16)
   02 REDEFINE ACCOUNT-NO
      03 DIVISION (A4)
      03 FILLER 1X
      03 GROUP (A5)
      03 FILLER 1X
      03 ENTITY (A5)
```

The following sample code refers to these field definitions:

```
Print .FieldDef("ROW-COUNT").LevelTypeTrail
' Prints "F"
Print .FieldDef("ROW").LevelTypeTrail
' Prints "S"
Print .FieldDef("ID").LevelTypeTrail
' Prints "SF"
Print .FieldDef("ACCOUNT-NO").LevelTypeTrail
' Prints "SF"
Print .FieldDef("DIVISION").LevelTypeTrail
' Prints "SRF"
Print .FieldDef(7).LevelTypeTrail
' Prints "SRX"
```

See Also

Level Property, page 356.

Name Property

This property gets the name of a Natural field in a data area definition.

Syntax

```
result = object.Name
```

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
result		string	Name of Natural field.

Rank Property

This property gets information about whether a field is a scalar field or part of an array, according to the following table:

Rank	Description
0	Scalar
1	1-dimensional array
2	2-dimensional array
3	3-dimensional array

Rank indicates the number of index values needed when reading or writing the field.

Syntax

```
result = object.Rank
```

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
result		Integer	Number of index values used when reading or writing the field.

See Also

DefinedRank Property, page 353.

Redefined Property

This property returns whether a field is redefined in a data area definition.

Syntax

```
result = object.Redefined
```

Part	Required	Data Type	Description
object	yes	NaturalField Def	Name of FieldDef object.
result		Boolean	If True, the field is redefined; if False, the field is not redefined.

Structure Property

This property gets a structure name if the specified field is part of a level 1 structure. If the specified field is not part of a level 1 structure, this property returns an empty string.

Syntax

```
result = object.Structure
```

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
result		string	Name of structure.

ThruIndex Property

This property gets the high index value for each dimension of an array field. Use this property in conjunction with the FromIndex property to determine the low and high index values.

Syntax

```
result = object.ThruIndex (index)
```

Part	Required	Data Type	Description
object	yes	NaturalFieldDef	Name of NaturalFieldDef object.
index	yes	Integer	Dimension number (1, 2, or 3).
result		Integer	High index value for the specified dimension of the array field.

Example

```
01 VALUES(N10/1:10,5:7)
With data.FieldDef("VALUES")
    For i = 1 To .Rank
        Print .FromIndex(i) & ":" & .ThruIndex(i)
    Next
End With
        Prints: 1:10
                5:7
```

See Also

DefinedRank Property, page 353, **Format Property**, page 353, and **Rank Property**, page 358.

NaturalFieldSpec Class

This class handles field specifications. It allows you to:

- Parse field specifications into individual components, such as the field name, structure, and indices

  or

- Create a field specification by defining the individual components and combining the components

The NaturalFieldSpec class has the following method and properties:

FieldName Property, page 361	**IndexType Property**, page 366
FieldSpec Property, page 362	**Indices Property**, page 367
IndexFrom Property, page 363	**ParseFieldSpec Method**, page 367
IndexOcc Property, page 364	**Structure Property**, page 368
IndexThru Property, page 365	

FieldName Property

This property gets or sets the name of a field in a field specification. It is read or write.

Syntax

```
result = object.FieldName
```

or

```
object.FieldName = value
```

Part	Required	Data Type	Description
object	yes	NaturalFieldSpec	Name of NaturalFieldSpec object.
value	yes	string	Field name to set.
result		string	Current field name.

See Also

ParseFieldSpec Method, page 367, and **FieldSpec Property**, page 362.

FieldSpec Property

This property determines the components of a field specification, groups the components into a field specification string, and gets the string. It is read-only.

Syntax

```
object.IndexFrom (index) = value
```

or

```
result = object.FieldSpec
```

Part	Required	Data Type	Description
object	yes	NaturalFieldSpec	Name of NaturalFieldSpec object.
result		string	Field specification string.

See Also

- **FieldName Property**, page 361
- **IndexFrom Property**, page 363
- **IndexOcc Property**, page 364
- **IndexThru Property**, page 365
- **IndexType Property**, page 366
- **Indices Property**, page 367
- **ParseFieldSpec Method**, page 367
- **Structure Property**, page 368

IndexFrom Property

This property gets or sets the low index value for each dimension of an array field in a field specification. Use this property in conjunction with the IndexThru property to determine the low and high index values.

Syntax

```
object.IndexFrom (index) = value
```

or

```
result = object.IndexFrom (index)
```

Part	Required	Data Type	Description
object	yes	NaturalFieldSpec	Name of NaturalFieldSpec object.
index	yes	Integer	Dimension number (1, 2, or 3).
result		Integer	Low index value for the specified dimension of the array field.

See Also

- **FieldSpec Property**, page 362
- **IndexOcc Property**, page 364
- **IndexThru Property**, page 365
- **IndexType Property**, page 366
- **Indices Property**, page 367
- **ParseFieldSpec Method**, page 367

IndexOcc Property

This property gets or sets the occurrence number of an array field in a field specification. It is read or write.

Syntax

object.IndexOcc(index) = *value*

or

result = *object*.IndexOcc(index)

Part	Required	Data Type	Description
object	yes	NaturalFieldSpec	Name of NaturalFieldSpec object.
index	yes	Integer	Index number for the occurrence.
value	yes	Integer	Occurrence number to set.
result		Integer	Current occurrence number.

Example

```
Dim nfs As New NaturalFieldSpec
With nfs
   .Name = "Row"
   .Indices = 1
   .IndexOcc(1) = 5
   Print .FieldSpec
End With
   Prints: Row(5)
```

See Also

- **FieldSpec Property**, page 362
- **IndexFrom Property**, page 363
- **IndexThru Property**, page 365
- **IndexType Property**, page 366
- **Indices Property**, page 367
- **ParseFieldSpec Method**, page 367

IndexThru Property

This property gets the high index value for each dimension of an array field in a field specification. Use this property in conjunction with the IndexFrom property to determine the low and high index values.

Syntax

```
object.IndexFrom (index) = value
```

or

```
result = object.IndexFrom (index)
```

Part	Required	Data Type	Description
object	yes	NaturalFieldSpec	Name of NaturalFieldSpec object.
index	yes	Integer	Dimension number (1, 2, or 3).
result		Integer	Low index value for the specified dimension of the array field.

See Also

- **FieldSpec Property**, page 362
- **IndexFrom Property**, page 363
- **IndexOcc Property**, page 364
- **IndexType Property**, page 366
- **Indices Property**, page 367
- **ParseFieldSpec Method**, page 367

IndexType Property

This property gets or sets the index type for an index to an array field in a field specification. It is read or write.

Syntax

object.IndexType(index) = *value*

or

result = *object*.IndexType(index)

Part	Required	Data Type	Description
object	yes	NaturalFieldSpec	Name of FieldSpec object.
index	yes	Integer	Index to the array field in the field specification.
value		Integer	Index type to set.
result		Integer	Current index type.

The index type can have one of the following values:

Type	Description	Example
1	Single occurrence number	Row(5)
2	Range	Row(1:5)
3	All occurrences	Row(*)

See Also

- **FieldSpec Property**, page 362
- **IndexFrom Property**, page 363
- **IndexOcc Property**, page 364
- **IndexThru Property**, page 365
- **Indices Property**, page 367
- **ParseFieldSpec Method**, page 367

Indices Property

This property gets or sets the number of indices for array fields in a field specification. It is read or write.

Syntax

object.Indices = *value*

or

result = *object*.Indices

Part	Required	Data Type	Description
object	yes	NaturalFieldSpec	Name of NaturalFieldSpec object.
value	yes	Integer	Indices to set (0, 1, 2, or 3).
result		Integer	Current indices (0, 1, 2, or 3).

See Also

- **FieldSpec Property**, page 362
- **IndexFrom Property**, page 363
- **IndexOcc Property**, page 364
- **IndexThru Property**, page 365
- **IndexType Property**, page 366
- **ParseFieldSpec Method**, page 367

ParseFieldSpec Method

This method separates a field specification into its individual components.

Syntax

object.ParseFieldSpec FieldSpec, (*index1*, *index2*, *index3*)

Part	Required	Data Type	Description
object	yes	NaturalFieldSpec	Name of NaturalFieldSpec object.
FieldSpec	yes	string	String containing the individual components of a field specification (for example, the field name, structure, and indices).

Part	Required	Data Type	Description (continued)
index1	no	Integer	Value of index1, if the field specification is for a field in an array.
index2	no	Integer	Value of index2, if the field specification is for a field in an array.
index3	no	Integer	Value of index3, if the field specification is for a field in an array.

See Also

FieldSpec Property, page 362, and **Structure Property**, page 368.

Structure Property

This property gets or sets a structure name if the specified field is part of a level 1 structure. It is read or write. If the field specification does not include a structure name, this property contains an empty string.

Syntax

```
object.Structure = value
```

or

```
result = object.Structure
```

Part	Required	Data Type	Description
object	yes	NaturalFieldSpec	Name of NaturalFieldSpec object.
value	yes	string	Name for the structure to set.
result		string	Current name of structure.

See Also

ParseFieldSpec Method, page 367, and **FieldSpec Property**, page 362.

UTILITY SUBROUTINES ON THE CLIENT

This chapter provides a brief description of each utility routine included in the Spectrum client framework. For full details, refer to the source code.

The following functions and subroutines are described in this chapter:

FixupRTF Function, page 384	**SetUppercaseStyle Subroutine**, page 400
GetPrivateProfileStringVB Function, page 385	**ValidAssignment Function**, page 401
GetWindowsDirectoryVB Function, page 386	

AppendSlash Function

This function appends a backslash (\) to the end of a directory name, if required. Use this function when a directory name may or may not end with a backslash and you want to combine it with a file name.

Syntax

```
result = AppendSlash(Str)
```

Part	Required	Data Type	Description
Str	yes	string	Input string.

Example

```
sfullpath = AppendSlash(sdir) & sfile
```

If sdir contains “C:\”, “C:\Files”, or “C:\Files\”, this expression evaluates to a properly formatted file name.

Data Type

String

ArrayDimensions Function

This function returns the number of dimensions in an array. You can also use this function to determine if a variable-sized array.

Syntax

```
Result = ArrayDimensions(Arr)
```

Part	Required	Data Type	Description
Arr	yes	variant	Array to be interrogated.

Example

```
Dim a1() As String
Dim a2(1 To 100, 1 To 3) As Variant
Dim a3() As String

ReDim a1(0 to 99)
Print ArrayDimensions(a1)    ' Prints 1.
Print ArrayDimensions(a2)    ' Prints 2.
Print ArrayDimensions(a3)    ' Prints 0.
```

Data Type

Integer

ASSERT Subroutine

This subroutine tests an assertion.

Syntax

```
ASSERT AssertCondition, Message
```

Part	Required	Data Type	Description
AssertCondition	yes	Boolean	Indicates whether assertion passed or failed.
Message	no	string	Message displayed if error occurs.

Example

```
' Class Stack
Public Sub Pop () As Variant
    ASSERT m_StackSize > 0, "Pop called on an "&_
                            "empty stack"
    ...
End Sub
```

CenterForm Subroutine

This subroutine centers a form relative to the screen or to another form.

Syntax

```
CenterForm frm, RelativeTo
```

Part	Required	Data Type	Description
frm	yes	form	Reference to the form to be centered.
RelativeTo	no	form	Reference to another form to be considered when centering the specified form. If not supplied, the form is centered relative to the screen.

Example

```
CenterForm Me            ' Centers on screen.
CenterForm Me, frmMain   ' Centers over main form.
```

CreateArray Function

This function creates and returns a 1-, 2-, or 3-dimensional array of variants.

Syntax

```
result = CreateArray(Low1, High1, Low2, High2, Low3, High3)
```

Part	Required	Data Type	Description
Low1	yes	long	Low bound for first array dimension.
High1	yes	long	High bound for first array dimension.
Low2	no	long	Low bound for second array dimension.
High2	no	long	High bound for second array dimension.
Low3	no	long	Low bound for third array dimension.
High3	no	long	High bound for third array dimension.

Example

```
Dim a1 As Variant

' Create a 2D array with indices 1 to 100, 1 to 10.
a1 = CreateArray(1, 100, 1, 10)
```

Data Type

Variant

CreateStringArray Function

This function is similar to the CreateArray function except it creates an array of strings.

Syntax

```
result = CreateStringArray(Low1, High1, Low2, High2, Low3, High3)
```

Part	Required	Data Type	Description
Low1	yes	long	Low bound for first array dimension.
High1	yes	long	High bound for first array dimension.
Low2	no	long	Low bound for second array dimension.
High2	no	long	High bound for second array dimension.
Low3	no	long	Low bound for third array dimension.
High3	no	long	High bound for third array dimension.

Example

```
Dim a1 As Variant

' Create a 2D array with indices 1 to 100, 1 to 10.
a1 = CreateStringArray(1, 100, 1, 10)
```

Data Type

Variant

See Also

CreateArray Function, page 374.

cstDisplayErrorTip Subroutine

This subroutine displays an error tip for a GUI control.

Syntax

```
cstDisplayErrorTip CurrForm, CurrControl, ErrColumn, _
                   Index1, Index2, Index3
```

Part	Required	Data Type	Description
CurrForm	yes	form	Reference to the form containing the control to which the error tip applies.
CurrControl	no	object	Reference to the control or TrueGridClass object to which the error tip applies (defaults to Screen.ActiveControl).
ErrColumn	no	integer	Grid column to which the error tip applies (TrueGridClass objects only).
Index1, 2, 3	no	integer	Unique index identifying the ObjectError object (only applicable for vector data). This value positions the error tip in a specific grid cell.

Example

```
Public Sub txt_Ordr_OrderAmount_GotFocus()
    ValueChanged = False
    cstSelectContents
    cstDisplayErrorTip Me
End Sub
```

See Also

Object Error Classes, page 247.

cstFormatMessage Function

This function formats a message in the CDPDA-M or CSASTD parameter data area by performing the message substitutions. It returns the result as a string.

Syntax

```
result = cstFormatMessage(Msg)
```

Part	Required	Data Type	Description
Msg	yes	NaturalDataArea	Data area containing fields used to create the message.

Example

```
disp.CallNat "SUBP1", parms, results, msg
If msg.Field("RETURN-CODE") <> "" Then
    MsgBox cstFormatMessage(msg), vbExclamation
End If
```

Data Type

String

cstRemoveAllErrors Subroutine

This subroutine removes all ObjectError objects from an ObjectErrors object for a form, hides the error tip, and updates the form's status bar to indicate that there are no errors on the form.

Syntax

```
cstRemoveAllErrors CurrForm
```

Part	Required	Data Type	Description
CurrForm	yes	form	Reference to the form from which errors are removed.

Example

```
cstRemoveAllErrors Me
```

Remarks

This subroutine assumes that the specified form has declared a variable named ObjErrors of type ObjectErrors.

See Also

Object Error Classes, page 247.

cstRemoveControlErrorsByIndex Subroutine

This subroutine removes all ObjectError objects from an ObjectErrors object for a form, GUI control (or TrueGridClass object), and index. It also updates the form's status bar to reflect the number of errors remaining on the form.

Syntax

```
cstRemoveControlErrorsByIndex CurrForm, CurrControl, _
                              Index1, Index2, Index3
```

Part	Required	Data Type	Description
CurrForm	yes	form	Reference to the form from which errors are removed.
CurrControl	yes	object	Reference to the GUI control or TrueGridClass object.
Index1, 2, 3	no	integer	Unique index identifying the ObjectError object (vector data only).

Example

```
If Action = ACTION_ROW_DELETE Then

    'remove validation errors for deleted grid cells
    cstRemoveControlErrorsByIndex Me, IncomeGrid, IncomeRow
End If
```

Remarks

Use this subroutine to remove ObjectError objects associated with a grid — especially when a user deletes a grid row. This subroutine assumes that the specified form has declared a variable named ObjErrors of type ObjectErrors.

See Also

Object Error Classes, page 247.

cstSelectContents Subroutine

This subroutine highlights the entire contents of a text box control by setting the SelStart and SelLength properties. To simulate the standard Windows behavior of selecting the text when you press Tab to access a field, call this subroutine in the GotFocus event for the text box.

Syntax

```
cstSelectContents ctl
```

Part	Required	Data Type	Description
ctl	no	TextBox	Reference to a text box control. If not supplied, the default is the active control on the screen.

Example

```
Private Sub txtCustomerNumber_GotFocus ()
    cstSelectContents
End Sub
```

cstSubst Function

This function substitutes values into a string marked with :n: substitution place holders. It returns the result as a string.

Syntax

```
result = cstSubst(Message, Values())
```

Part	Required	Data Type	Description
Message	yes	string	Message containing substitution place holders.
Values()	yes	string array	Variable size array of strings containing substitution values.

Example

```
Print cstSubst(":1:record(s) displayed", icount)
' Prints "1 record(s) displayed" if icount = 1.
```

Data Type

String

FileExists Function

This function tests if a file exists by attempting to open the file. If the file exists, this function returns True; if the file does not exist, this function returns False.

Syntax

```
result = FileExists(FileName)
```

Part	Required	Data Type	Description
FileName	yes	string	Name of the file and, optionally, the path.

Example

```
If FileExists(sfilename) Then
    ' Process file.
    Open sfilename For Binary Access Read ...
End If
```

Data Type

Boolean

FindFirst Function

This function searches a string for the first occurrence of a character in a set of characters. It returns the position and character found.

Syntax

```
result = FindFirst(StartPos, SearchText, SearchChars, FoundChar)
```

Part	Required	Data Type	Description
StartPos	yes	long	Character position within the string at which the search begins.
SearchText	yes	string	Text to be searched.
SearchChars	yes	string	Set of characters to be found.
FoundChar	yes	string	Returns the character found. This argument is used for output only (passed ByRef).

Example

```
' Search for first digit starting at position 2.
lpos = FindFirst(2, sfilename, "0123456789", _
                 sfoundchar)
If lpos > 0 Then
    ' sfoundchar contains the character found.
End If
```

Data Type

Long

FixupRTF Function

This function changes any embedded backslash characters (\) in a string to two backslash characters so the string can be displayed properly in a RichTextBox control.

Syntax

```
result = FixupRTF(Text)
```

Part	Required	Data Type	Description
Text	yes	string	String to be interrogated for backslashes.

Example

```
RichTextBox1.SelRTF = FixupRTF(sinsertstring)
```

Data Type

String

GetPrivateProfileStringVB Function

This function reads a string in a Windows .INI file. It is a Visual Basic wrapper for the GetPrivateProfileString function in the Win32 API.

Syntax

```
result = GetPrivateProfileStringVB(Section, Key, Default, FileName)
```

Part	Required	Data Type	Description
Section	yes	string	Section within the .INI file to be interrogated.
Key	yes	string	Key within the specified section of the .INI file to be interrogated.
Default	yes	string	Value returned if the function is unable to find a value for the specified Section and Key.
FileName	yes	string	Name of the .INI file to be used.

Example

```
sdisp = GetPrivateProfileStringVB( _
            "SDC", _
            "DefaultDispatcher", _
            "", _
            sinifilename)
```

Data Type

String

GetWindowsDirectoryVB Function

This function returns the name of the Windows directory. It is a Visual Basic wrapper for the GetWindowsDirectory function in the Win32 API.

Syntax

```
result = GetWindowsDirectoryVB()
```

Example

```
swindir = GetWindowsDirectoryVB()
```

Data Type

String

HideErrorTip Subroutine

This subroutine hides an error tip.

Syntax

```
HideErrorTip()
```

Example

```
Public Sub txt_Ordr_OrderNumber_LostFocus()
    Dim Value As String

    HideErrorTip
    …
End Sub
```

IsForegroundApplication Function

This function determines whether a form is currently the foreground application in Windows. If the current application is the foreground application, this function returns True. If it is not the foreground application, this function returns False.

Syntax

```
result = IsForegroundApplication()
```

Remarks

Use this function when you want to execute code only if the application is currently active, such as code in a timer event that updates the screen.

Example

```
'Skip this processing if the application is not the foreground
'application
If Not IsForegroundApplication() Then
    Exit Sub
End If

'Do the processing.
```

Data Type

Boolean

IsMDIChild Function

This function determines whether a form is an MDI (multiple-document interface) child window. If the specified form is an MDI child window, this function returns True. If the specified form is the MDI frame window or has its MDIChild property set to False, this function returns False.

Syntax

```
result = IsMDIChild(frm)
```

Part	Required	Data Type	Description
frm	yes	form	Reference to form to be interrogated.

Example

```
' Do special processing for all MDI child forms
' currently loaded.
For Each frm in Forms
    If IsMDIChild(frm) Then
        ...
    End If
Next
```

Data Type

Boolean

Max Function

This function returns the maximum of two values. It works with any data type.

Syntax

```
result = Max(Value1, Value2)
```

Part	Required	Data Type	Description
Value1	yes	variant	First value to compare.
Value2	yes	variant	Second value to compare.

Example

```
' Don't let the width drop below 50.
iwidth = Max(iwidth, 50)
```

Data Type

Variant

Min Function

This function returns the minimum of two values. It works with any data type.

Syntax

```
return = Min(Value1, Value2)
```

Part	Required	Data Type	Description
Value1	yes	variant	First value to compare.
Value2	yes	variant	Second value to compare.

Example

```
' Don't let the width exceed 250.
iwidth = Min(iwidth, 250)
```

Data Type

Variant

MoveFormSafely Subroutine

This subroutine moves a non-MDI child form to a new location on the screen, ensuring that the entire form is displayed on the screen.

Syntax

```
MoveFormSafely frm,NewLeft,NewTop
```

Part	Required	Data Type	Description
frm	yes	form	Reference to form to be considered.
NewLeft	yes	integer	Left coordinate for the form. The function updates the parameter to reflect the form's new left coordinate.
NewTop	yes	integer	Top coordinate for the form. The function updates the parameter to reflect the form's new top coordinate.

Example

```
' inewleft or inewtop could position the form so
' that part of it is offscreen. However
' MoveFormSafely ensures that this doesn't happen.
MoveFormSafely Me, inewleft, inewtop
```

PadLeft Function

This function pads a string on the left with spaces or a specified character to a specified width.

Syntax

```
result = PadLeft(Value, Length, PadChar)
```

Part	Required	Data Type	Description
Value	yes	variant	Value to be padded with characters.
Length	yes	integer	Number of characters to pad.
PadChar	no	variant	Either a character, integer (representing an ASCII or UNICODE key value), or string value. If it is a string value, this function uses the first character. If this argument is not specified, the default is a blank.

Example

```
' Pad with spaces to a length of 10 characters.
stext = PadLeft(stext, 10)
' Pad with zeroes.
stext = PadLeft(stext, 10, "0")
```

Data Type

String

PadRight Function

This function pads a string on the right with spaces or a specified character to a specified width.

Syntax

```
result = PadRight(Value, Length, PadChar)
```

Part	Required	Data Type	Description
Value	yes	variant	Value to be padded with characters.
Length	yes	integer	Number of characters to pad.
PadChar	no	variant	Either a character, integer (representing an ASCII or UNICODE key value), or string value. If it is a string value, this function uses the first character. If this argument is not specified, the default is a blank.

Example

```
' Pad with spaces to a length of 10 characters.
stext = PadRight(stext, 10)
' Pad with periods.
stext = PadRight(stext, 10, ".")
```

Data Type

String

ParseErrorString Subroutine

This subroutine parses a string containing an error message, error number, and error source into three separate variables.

Syntax

```
ParseErrorString ErrorMsg, ErrorNr, ErrorSrc
```

Part	Required	Data Type	Description
ErrorMsg	yes	string	Message to be parsed (on return, the error message).
ErrorNr	yes	long	Returns the error number in ErrorMsg.
ErrorSrc	yes	integer	Returns the error source in ErrorMsg.

Example

```
Dim ErrorMsg As String
Dim ErrorNr As Long
Dim ErrorSrc As Integer

ErrorMsg = "SRC=" & ERROR_SOURCE_BDT & vbTab & _
           "NUM=" & 1080 & vbTab & _
           "MSG=" & "You made an error"
ParseErrorString ErrorMsg, ErrorNr, ErrorSrc
Debug.Print ErrorMsg     'prints: You made an error
Debug.Print ErrorNr      'prints: 1080
Debug.Print ErrorSrc     'prints: 4
```

Remarks

Errors returned by the ValidAssignment function compress the error message, source, and number into a single string argument using the MSG, SRC, and NUM delimiters. Use this subroutine to separate the error components before calling the SetObjectError subroutine. The ErrorSrc argument returns an integer value corresponding to one of the values specified by the constants prefixed with ERROR_SOURCE*. Use these constants for comparison purposes. If the input ErrorMsg argument does not follow the convention described above, this subroutine assumes the contents of the ErrorMsg argument pertain only to the error message and sets the ErrorNr and ErrorSrc values to 0.

See Also

ValidAssignment Function, page 401, and **SetObjectError Subroutine**, page 399.

QueryDeletion Function

This function displays a dialog to confirm a deletion request from a user.

Syntax

```
result = QueryDeletion(FormCaption, ObjectDesc, KeyValue)
```

Part	Required	Data Type	Description
FormCaption	yes	string	Caption for confirmation dialog.
ObjectDesc	yes	string	Type of business object (customer or order, for example) requesting the deletion function. This argument is included in the confirmation message.
KeyValue	yes	string	Business object. If this argument is not blank, it is included in the confirmation message.

Example

```
If Action = ACTION_DELETE And gConfirmDeletion Then
    ModResponse = QueryDeletion(FormCaption, ObjectDesc, Me.Caption)
    If ModResponse = vbNo Then GoTo ExitSub
End If
```

Remarks

This function returns an integer value indicating the user's response. Use the vbYes and vbNo constants to determine how to act on the return value.

Data Type

Integer

QueryModifications Function

This function displays a dialog confirming whether changes to the current business object are saved before proceeding.

Syntax

```
result = QueryModification(FormCaption, ObjectDesc, KeyValue, Action)
```

Part	Required	Data Type	Description
FormCaption	yes	string	Caption for confirmation dialog.
ObjectDesc	yes	string	Type of business object (customer or order, for example) requesting the deletion function. This argument is included in the confirmation message.
KeyValue	yes	string	Business object. If this argument is not blank, it is included in the confirmation message.
Action	yes	integer	Indicates whether the save function is implemented as an add or update. Use the ACTION_ADD or ACTION_UPDATE constants.

Example

```
If InternalObject.KeyChanged Then
    ModResponse = QueryModifications(FormCaption, ObjectDesc, _
                                     Me.Caption, ACTION_ADD)
Else
   ModResponse = QueryModifications(FormCaption, ObjectDesc, _
                                    Me.Caption, ACTION_UPDATE)
End If
```

Remarks

This function returns an integer value indicating the user's response. Use the vbYes and vbNo constants to determine how to act on the return value.

Data Type

Integer

RemoveUnneededControlErrors Subroutine

This subroutine removes unneeded ObjectError objects associated with a form and GUI control.

Syntax

```
RemoveUnneededControlErrors CurrForm, CurrControl, _
                            ValueChanged, ErrColumn, _
                            Index1, Index2, Index3
```

Part	Required	Data Type	Description
CurrForm	yes	form	Reference to the form from which errors are removed.
CurrControl	yes	object	Reference to a GUI control or TrueGridClass object.
ValueChanged	yes	Boolean	Indicates whether this subroutine is called after a change is made to the contents of a GUI control.
ErrColumn	no	integer	Grid column from which errors are removed (TrueGridClass objects only).
Index1, 2, 3	no	integer	Unique index identifying the ObjectError object (vector data only).

Example

```
RemoveUnneededControlErrors Me, IncomeGrid, ValueChanged, _
                            ColIndex, IncomeRow
```

Remarks

This subroutine is typically called from a Lost_Focus event. If the contents of the GUI control change (as indicated by the ValueChanged argument), all ObjectError objects associated with the GUI control are removed.

If the contents of the GUI control do not change, only local ObjectError objects (those with a MsgType of ERROR_SOURCE_SDC, ERROR_SOURCE_BDT, or ERROR_SOURCE_VALIDATE) are removed. These local validations are reapplied in a Lost_Focus event, whereas validations performed on the server are not applied until the user makes a specific request (such as update or add). Using the ValueChanged argument ensures that the ObjectError objects originating on the server are preserved if the user only tabs through the GUI controls on a form.

See Also

Object Error Classes, page 247.

ResizeForm Subroutine

This subroutine resizes the client area on a form to a specified size.

Syntax

```
ResizeForm frm, NewWidth, NewHeight
```

Part	Required	Data Type	Description
frm	yes	form	Reference to the form to be resized.
NewWidth	yes	integer	New width for the form.
NewHeight	yes	integer	New height for the form.

Remarks

Call this procedure to resize a form when you know how large the client area must be. Set the ScaleMode of the form to either vbTwips (1) or vbPixels (3). You must pass the width and height as parameters. If you only want to set the width or height and leave the other property alone, pass –1 (negative one) for the parameter you do not want to change.

Example

```
' Make the form wide enough to accommodate the
' Cancel button plus a padding of 8 pixels.
ResizeForm Me, _
           cmdCancel.Left + cmdCancel.Width + 8, _
           -1
' Resize the form to accommodate the ListBox
' control.
With lstModules
    ResizeForm Me, .Left + .Width + 8, _
                   .Top + .Height + 8
End With
```

SetObjectError Subroutine

This subroutine adds a new ObjectError object to the ObjectErrors object for a form. The ObjectError object is associated with a GUI control or a TrueGridClass object.

Syntax

```
SetObjectError p_ErrForm, p_ErrControl, p_ErrMsgNr, _
               p_ErrMsg, p_ErrMsgType, p_ErrColumn, _
               p_Index1, p_Index2, p_Index3 As Variant
```

Part	Required	Data Type	Description
p_ErrForm	yes	form	Reference to a form containing a reference to an ObjectErrors object.
p_ErrControl	yes	object	Reference to a GUI control or TrueGridClass object.
p_ErrMsgNr	yes	long	Number which, in conjunction with a message type, uniquely identifies an error.
p_ErrMsg	yes	string	Message describing the nature of the error.
p_ErrMsgType	yes	integer	Source of the error. Use the constants prefixed with ERROR_SOURCE* as valid error sources.
p_ErrColumn	no	integer	Errors associated with a TrueGridClass object; identifies the column in error.
p_Index1, 2, 3	no	integer	Unique index identifying the ObjectError object (only applicable for vector data).

Example

```
If ErrorMsg <> "" Then
    ParseErrorString ErrorMsg, ErrorNr, ErrorSrc
    SetObjectError Me, txt_OrdM_OrderNumber, ErrorNr, _
                   ErrorMsg, ErrorSrc
End If
```

See Also

Object Error Classes, page 247.

SetUppercaseStyle Subroutine

This subroutine sets the Windows style bit for a text box control so the control converts all text to upper case.

Syntax

```
SetUppercaseStyle ctl
```

Part	Required	Data Type	Description
ctl	yes	Control	Name of the text box control.

Example

```
SetUppercaseStyle txtUserID
SetUppercaseStyle txtPassword
```

ValidAssignment Function

This function assigns a value to a field in a NaturalDataArea object for a Visual Basic maintenance dialog. It converts the value from a display format to an underlying data format (BDT logic), invokes the local business validation logic in a Visual Basic maintenance object, and then invokes logic in the maintenance object to assign the value to a NaturalDataArea field.

Syntax

```
result = ValidAssignment(Value, DataArea, FieldSpec, _
                         ErrorMessage, TypeName, _
                         NatFormatLength As Variant, _
                         ConsiderBDT As Variant)
```

Part	Required	Data Type	Description
Value	yes	variant	Value assigned to the NaturalDataArea field.
DataArea	yes	object	Reference to either the NaturalDataArea object or Visual Basic maintenance object.
FieldSpec	yes	string	Name of the NaturalDataArea field, including any indices.
ErrorMessage	yes	string	ByRef argument used to return an error message if a validation error occurs.
TypeName	no	string	Business data type (BDT) associated with this value. Use a constant prefixed with BDT* to identify the BDT.
NatFormatLength	no	string	Natural format and length of the underlying NaturalDataArea field. If this argument is not supplied, this function uses the format and length defined in the FieldSpec object.
ConsiderBDT	no	Boolean	Indicates whether BDT conversion logic is applied as part of the assignment process. If this argument is not supplied, the default is True.

Example

```
ValidAssignment Value, InternalObject, "ORDER-NUMBER", _
                ErrorMsg, NatFormatLength:="N6"
txt_OrdM_OrderNumber.Text = Value
If ErrorMsg <> "" Then
    ParseErrorString ErrorMsg, ErrorNr, ErrorSrc
    SetObjectError Me, txt_OrdM_OrderNumber, ErrorNr, _
                   ErrorMsg, ErrorSrc
End If
```

Remarks

If a validation error occurs in any of the steps performed by this function:

- Control returns immediately to the caller
- The function returns False
- An error message is returned to the caller through the ErrorMsg argument
- The underlying NaturalDataArea field is not updated

Data Type

Boolean

See Also

- **ParseErrorString Subroutine**, page 394
- **SetObjectError Subroutine**, page 399
- **Object Error Classes**, page 247
- **Business Data Type (BDT) Classes**, page 155

SPECTRUM DISPATCH CLIENT INTERNALS

This chapter describes how Construct Spectrum converts Natural data to Visual Basic data and vice versa for transmission between a server and a client.

The following topics are covered:

Conversions When You Assign Data to a Field

The Spectrum Dispatch Client (SDC) converts Natural data types (A, B, C, D, F, I, L, N, P, and T) to Visual Basic data types (Integer, Long, and String) and vice versa. These conversions occur any time you read or write a field value in a NaturalDataArea object.

The conversions occur because NaturalDataArea objects store field values internally using byte sequences that are almost identical to the byte sequences Natural uses in real Natural variables. However, NaturalDataArea objects expose field values as Visual Basic variants for the convenience of the Visual Basic programmer.

Thus variants must be converted to Natural data types when writing a field, and Natural data types must be converted to variants when reading a field.

The following table shows how different variant subtypes are converted or coerced into values that can be stored in a Natural field. These conversions/coercions happen when you use the Field property or the SetField method to assign data to a field. The number in each cell of the table specifies additional conditions that must be met for the conversion to take place. The letter in each cell of the table specifies conversions or coercions that are applied to the variant value. They are listed on the following page.

Note: All other variant subtypes not in this table (String Array, for example) cause an error.

	A	Bm	C	D	F	I	L	Nm.n	Pm.n	T
Empty	1a	1n	1n	1j	1g	1g	1d	1g	1g	1j
Null	1a	1n	1n	1j	1g	1g	1d	1g	1g	1j
Integer	1c	-	-	1q	1	3	1e	2	2	1s
Long	1c	-	-	5q	1r	3	1e	2	2	5s
Currency	1c	-	-	5q	1r	3h	1e	2f	2f	5s
Single	1c	-	-	5q	4r	3h	1e	2f	2f	5s
Double	1c	-	-	5q	4r	3h	1e	2f	2f	5s
Date	5t	-	-	5k	-	-	-	-	-	5
String	1b	1b	-	-	-	-	lw	-	-	-
Boolean	1m	-	-	-	1u	1u	1	1u	1u	-
Byte	1c	-	-	-	1	3	1e	2	2	-
Byte Array	6v	6p	1p	-	-	-	-	-	-	-

Conversion of Variant Subtypes

Additional conditions that must be met:

1 Will fit.

2 The number of digits in the whole number portion must be no more than m.

3 The value must be in the range of possible values.

4 For an F4, the value must be in the range of the Visual Basic Single data type; for an F8, the value must be in the range of the Visual Basic Double data type.

5 The value must be in Natural's date range 1582.01.01 (Variant value -116145) to 2699.12.31 (Variant value 284160).

6 The byte array must have one dimension.

The conversion rules are:

Conversion Rule	Description
a.	Becomes an empty string.
b.	Truncated to fit or padded with spaces.
c.	Converted to a string format, left justified, and truncated to fit or padded with spaces.
d.	Becomes False.
e.	Any non-zero value becomes True, otherwise False.
f.	If the value has more decimal places than n, it is rounded to n.
g.	Becomes 0.
h.	Any fractional portion is dropped.
j.	Becomes an uninstalled date or time.
k.	If the value contains a time portion, it is discarded.
m.	True becomes X, False becomes an empty string.
n.	Fills the field with the byte value 0.
p.	If the byte array is shorter than the field, the field is padded with byte value 0. If the array is longer, excess bytes are discarded.
q.	Any fractional portion is discarded and Visual Basic's mapping of numbers to dates is used. For example, 0 becomes 1899.12.30, and 36526 becomes 2000.01.01.
r.	For an F4, 7 significant digits are maintained; for an F8, 16 digits are maintained if the exponent is between -99 and 99, or 15 digits if the exponent is less than -99 or greater than 99.
s.	Visual Basic's mapping of numbers to a date and time is used.
t.	Converted to a string date format (according to the date format selected), left justified, and truncated to fit or padded with spaces.
u.	True becomes 1, False becomes 0.
v.	If the byte array is shorter than the field, the field is padded with spaces. If the array is longer, excess bytes are discarded.
w.	An empty string or a string filled with spaces becomes False. Everything else becomes True.

Variant Subtypes Returned From Fields

The variant subtype returned by the Field property and GetField method varies, depending on the Natural format and length and/or the value in the field. The following table describes these conditions:

Format	Conditions	Variant Subtype
A		String
B		Byte array
C		Byte array
D	Field contains: • an uninitialized date • any other value	 • Null • Date
Fm	• m = 4 • m = 8	• Single • Double
Im	• m = 1 • m = 2 • m = 4	• Integer • Integer • Long
L		Boolean
Nm.n	n = 0 and 1<=m<=4	Integer
Pm.n	• n = 0 and 5<=m<=9 • any other combination	• Long • Double
T	Field contains: • an uninitialized time • any other value	 • Null • Time

Transmission Representation

The representation of field values transmitted between the client and the server depends on the format of the field and is summarized in the following table:

Format/Length	Wire Encoding
Am	Sequence of m characters. If the value in the field is less than m characters, it is padded with spaces on the end.
Bm	Sequence of hex digits, where each group of two hex digits forms one byte.
C	Sequence of four hex digits, where each group of two hex digits forms one byte.
D	Sequence of 8 digits in YYYYMMDD format. If the field contains an uninitialized date, the transmission representation is a sequence of 8 ASCII zeros.
F4	Sequence of characters in the format: `+0.000000E+00`

F8	If the exponent is between –99 and +99 inclusive, a sequence of characters in the format:
	`+0.000000000000000E+00`
	If the exponent is less than –99 or greater than +99, a sequence of characters in the format:
	`+0.00000000000000E+000`

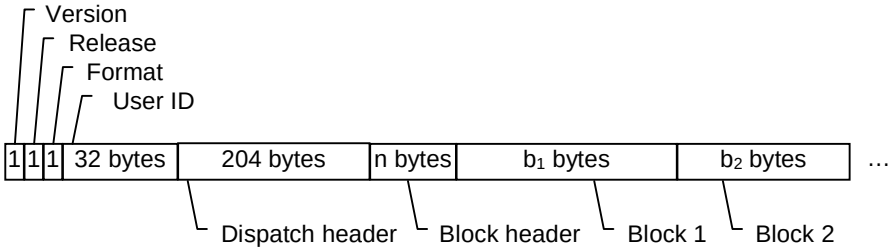
I1	Explicit leading sign character (+ or –) followed by a sequence of 3 digits.
I2	Explicit leading sign character (+ or –) followed by a sequence of 5 digits.
I4	Explicit leading sign character (+ or –) followed by a sequence of 10 digits.
L	0 for False and 1 for True.
Nm.n or Pm.n	If *n* = 0, explicit leading sign character (+ or –) followed by a sequence of *m* digits. If *n* > 0, a sequence of *m* + *n* digits with an explicit decimal character after the *m*th digit.
T	• uninitialized time: 000000000000000 • date only: YYYYMMDD0000000 • time only: 00000000HHIISST • both date and time: YYYYMMDDHHIISST

Understanding the Transmission Protocol

When debugging communications between the client and the Spectrum dispatch service, it is sometimes necessary to examine the transmitted data at the byte level to resolve problems. This section describes how the request data and response data are transmitted.

Request Data

The request data is composed of the following components:



Version and Release are each a single byte and must have the value 1.

Format is a single byte indicating whether the request data is ASCII or EBCDIC, compressed or not, and encrypted or not. The following table describes the bits in Format:

Bit	Description
0	0 = ASCII 1 = EBCDIC
1	0 = not compressed 1 = compressed
2	0 = not encrypted 1 = encrypted
3	0 = secured dispatch header 1 = unsecured dispatch header
4 to 7	Reserved

The user ID and dispatch header are always transmitted in the EBCDIC character set. Furthermore, if security is enabled for the dispatch service, the dispatch header is also encrypted. The block header and the blocks are always translated, compressed, and/or encrypted, based on the values in the format byte.

The dispatch header identifies the domain, object, version, method, user language, and trace options. The Spectrum dispatch service uses these values to look up the subprogram proxy and steplib chain. If the server component is secured, the dispatch service also uses these values, along with the public key in the reserved area, to authenticate the user and the request.

The block header consists of one byte for each level 1 field or structure in the parameter data of the subprogram being called. Each byte can be either 0 or 1, where 0 indicates the block is not in the request data and 1 indicates the block is in the data.

The individual blocks follow the block header. The size of each block depends on the fields within the block. The bytes in each block can be determined by reading the PackedData property of the NaturalDataArea object corresponding to each block.

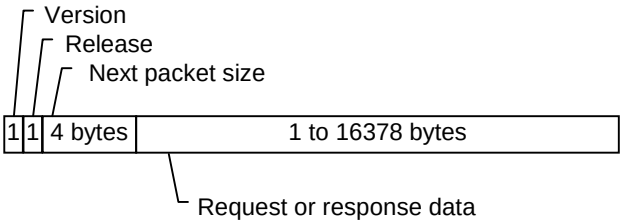
Once the request data has been built, it is processed further depending on the values of the Compress and Encrypt properties, and the PreferredCharSet setting in the dispatch service settings. The request data is first converted to EBCDIC if PreferredCharSet is set to EBCDIC. Next, the data is compressed if Compress is set to True. Finally, it is encrypted if Encrypt is set to True.

The resulting data is transmitted to the Spectrum dispatch service via EntireX Communicator.

Packetized Transmission Protocol

The size of the request data or response data may be larger than the maximum amount of data that can be sent in a single EntireX Communicator message. Therefore, the data is divided into messages no larger than 16384 bytes. (This corresponds to four full long message buffers in EntireX Communicator. For more information, refer to the EntireX Communicator documentation.) Each message is called a packet and contains a header and a portion of the data. This packetized transmission protocol is used when sending data from the client to the server and vice versa.

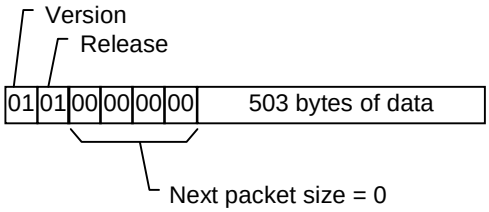
Each packet has the following format:



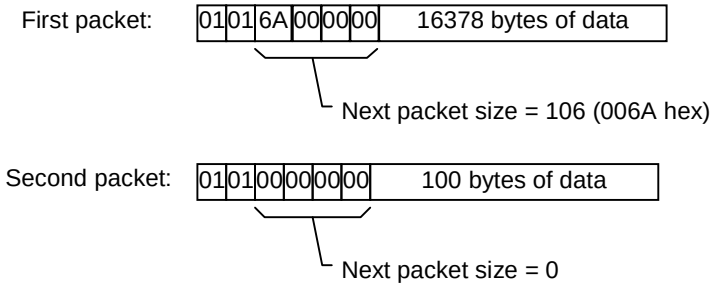
Version and Release are each a single byte and must have the value 1. Next packet size indicates the total length of the next packet. In the last packet, this field has the value 0. This field is encoded as a 32-bit integer in little-endian format, where the least significant byte is first and the most significant byte is last.

Up to 16378 bytes of request or response data follow. Just as the sender of the data must break the data up into packets, so must the receiver of the data reassemble those packets to obtain the original data. The following examples illustrate how the packetized transmission protocol works.

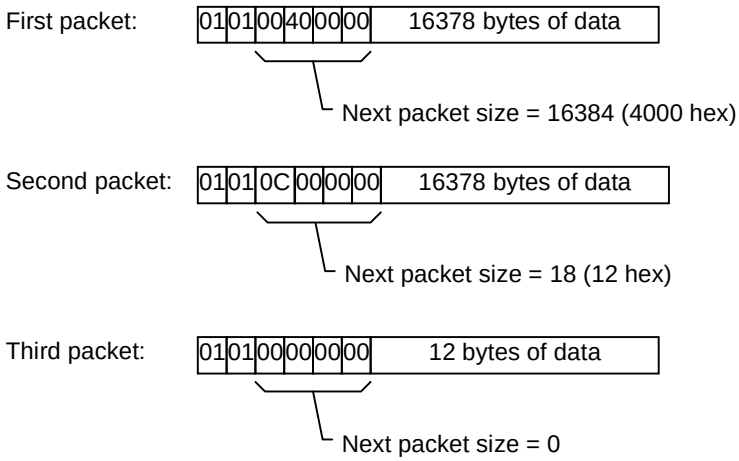
For example, assume the request data is 503 bytes long (35 byte request header + 204 byte dispatch header + 264 bytes of block header and blocks). This requires one packet containing the following bytes:



Now assume the request data is 16478 bytes long. This requires two packets (16378 bytes in the first, 100 bytes in the second) containing the following bytes:

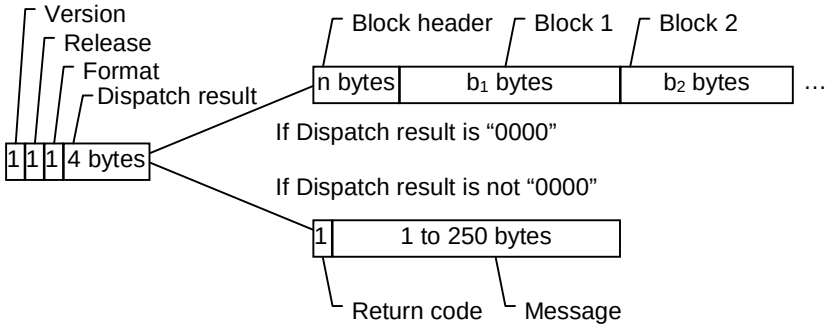


Finally, if the request data is 32768 bytes long, this requires three packets (16378 bytes, 16378 bytes, and 12 bytes):



Response Data

The response data is made up of the following components:



Version and Release are each a single byte and must have the value 1. Format is a single byte that has the same interpretation as in the request data. All data, including the Dispatch result and everything following it, is compressed and/or encrypted, as indicated in the format byte. The translation bit indicates the character set of the data being sent. The first three bytes of the response data are always binary and are never translated, compressed, or encrypted.

The Dispatch result is a 4-character message number that contains an error code from the Spectrum dispatch service. If the error code is all zeros, no error occurred in the Spectrum dispatch service or any of its subsystems. In this case, the data that follows the dispatch result is the block header and the individual blocks, having the same format as in the request data.

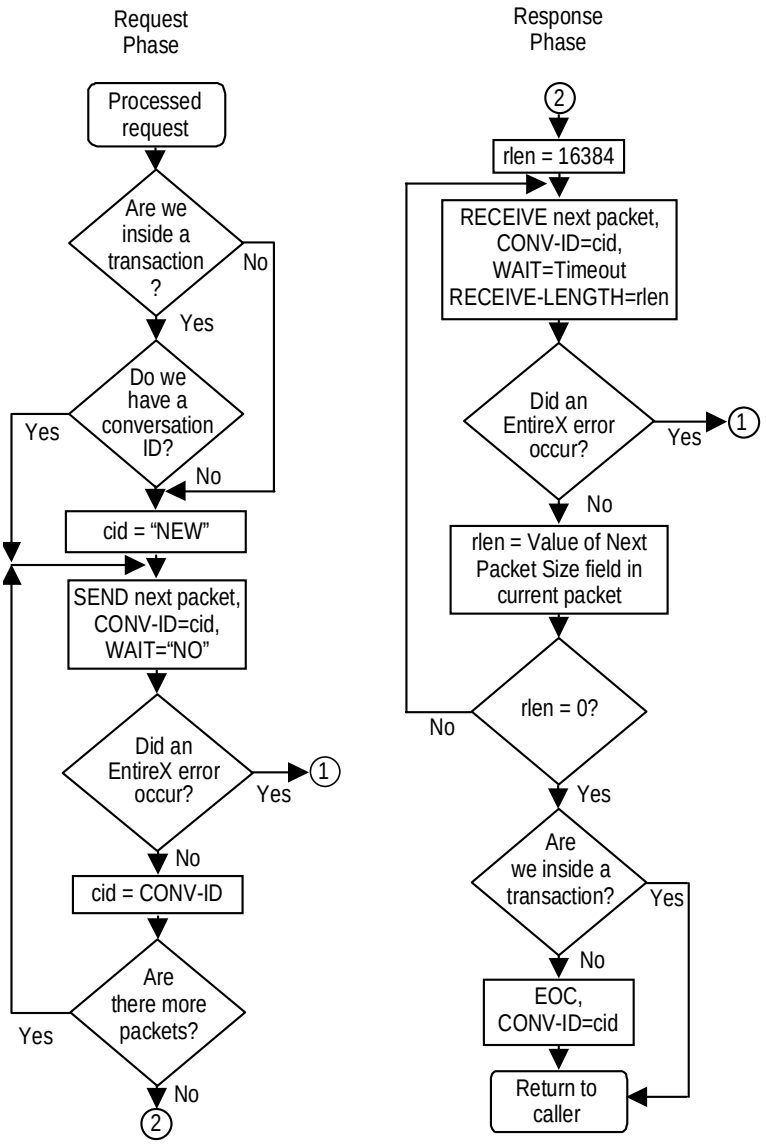
If the dispatch result is not all zeros, an error occurred in the Spectrum dispatch service or one of its subsystems, and the dispatch result is the error code. In this case, the return code indicates which subsystem caused the error, and Message is a textual message that can be displayed to the end-user. The Spectrum dispatch client concatenates the return code and the dispatch result and returns them in the ErrorNumber property.

The following table shows all possible return code values:

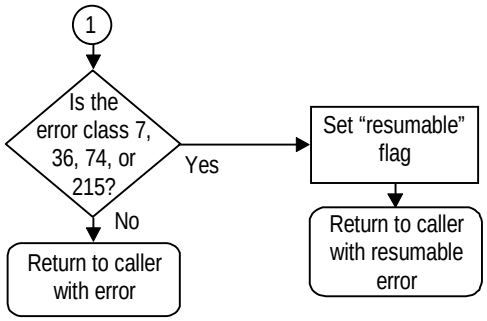
Return Code	Description
C	Error in the communications protocol.
D	Any error in the Dispatch Service.
I	Error in the subprogram proxy.
N	Natural runtime error.
S	Application security errors.
U	Error is related to looking up the user.

How EntireX Communicator Calls Are Used

When debugging communications between the client and the Spectrum dispatch service, it is sometimes necessary to understand the EntireX Communicator calls that send the request data to the server and receive the response data from the server. The following flowcharts illustrate this process.



EntireX Communicator Calls in the Request-Response Cycle



EntireX Communicator Error Handling