Reference Guide

RightFielder Object

32/64 BIT

Multiplatform

MELISSA DATA®

RightFielder Object

Reference Guide

Melissa Data Corporation

Copyright

Trademarks

Melissa Data Corporation

22382 Avenida Empresa
Rancho Santa Margarita, CA 92688-2112

Phone: 1-800-MELISSA (1-800-635-4772)
Fax: 949-589-5211

E-mail: info@MelissaData.com
Internet: www.MelissaData.com

For the most recent version of this document, visit
http://www.melissadata.com/

Document Code: DQTAPIRFRG
Revision Number: 04102013.14

Dear Developer,

I would like to take this opportunity to thank you for your interest in Melissa Data products and introduce you to the company.

Melissa Data has been a leading provider of data quality and address management solutions since 1985. Our data quality software, Cloud services, and data integration components verify, standardize, consolidate, enhance and update U.S., Canadian, and global contact data, including addresses, phone numbers, and email addresses, for improved communications and ROI. More than 5,000 companies rely on Melissa Data to gain and maintain a single, accurate and trusted view of critical information assets.

This manual will guide you through the functions of our easy-to-use programming tools. Your feedback is important to me, so please don't hesitate to email your comments or suggestions to me at: Ray@MelissaData.com.

I look forward to hearing from you.

Best Wishes,

Raymond F. Melissa

President/CEO

Table of Contents

Chapter 1
Introduction to RightFielder Object

RightFielder Object's powerful entity recognition and identification algorithms allow extraction of contact information from freeform or unfielded textual data. This product will identify, parse and reorganize input data into usable data types, assuring that even the most inconsistent data entry will be properly validated and stored.

It can parse a single line of input text containing many datatypes - whether or not it contains a delimiter.

RightFielder Object can also:

- Identify run-on fields where data entry was continued onto the next field.
- Reorganize badly fielded databases. This situation often occurs when records have varying number of contact names, address lines, or contain City, State and ZIP information as either a single line of text or in separated fields.
- Verify real-time data entry or web form data where data was entered in the wrong field, or all entered in a single field.
- Prevent costly errors when verifying addresses, presorting a mailing list, or running queries against your database for a particular data type.
- RightFielder Object can also be used to clean up existing databases where data entry or appending records has made field usage inconsistent.

The Parsing Engine

RightFielder Object uses different algorithms to parse your data. By default, RightFielder Object recognizes the tab character, pipe characters, and new lines as 'hard' delimiters. These hard delimiters tell RightFielder where a single piece of data starts and stops, greatly increasing the accuracy of processing. If no hard delimiters exist (run-on data types), the Object will use additional algorithms to best decide how to identify and parse the data into individual output properties.

Supplying more input data to RightFielder Object will increase the chance that it will accurately identify each piece of data correctly - especially if you supply a valid address, city, state and zip. In other words, if you pass in a single piece of data, RightFielder may place it in a questionable output because it can represent many different data types.

For example, an input string that consists only of "Anywhere CO" might be output as a city named "Anywhere" in the state of Colorado. If the input string, "John Smith, Anywhere CO, Anytown MA 01234," were used, then RightFielder Object would now correctly identify "Anywhere CO" as a company name.

1.1: License String

When you purchase a license to use RightFielder Object or download the trial version, you will be given a license string, a series of characters that activates RightFielder Object for the length of your subscription (or 30 days, in the case of the trial version).

There are two ways to use the license string with RightFielder Object.

Environment Variable: MD_LICENSE

It is preferable to set the license string into an environment variable called MD_LICENSE. Using an environment variable makes it much easier to update the license string after renewing your subscription without having to edit your code and re-compile the application.

Windows

Windows users can set environment variables by doing the following:

1. Select *Start* > *Settings*, and then click **Control Panel**.
2. Double-click **System**, and then click the *Advanced* tab.
3. Click **Environment Variables**, and then select either *System Variables* or *Variables* for the user X.
4. Click **New**.
5. Enter "MD_LICENSE" in the **Variable Name** box.

6 Enter the license string in the **Variable Value** box and then click **OK**.

Please remember that these settings take effect only upon start of the program. It may be necessary to quit and restart the development environment to incorporate the changes.

Linux/Solaris/HP-UX/AIX

Unix-based OS users can simply set the license string via the following entry (use the actual license string instead):

```
export MD_LICENSE=A1B2C3D4E5
```

If this setting is placed in the `.profile`, remember to restart the shell.

Programatically

The other method of setting the license string is to use the **SetLicenseString** function described on page 8. This is normally used for testing purposes. For actual deployment of applications developed with RightFielder Object, using the environment variable is the preferred method.

1.2: Using RightFielder Object

1 Create an instance of RightFielder Object.

```
Create RightFielderPtr as New Instance of mdRightFielder.
```

2 Initialize the data files.

```
CALL SetPathToRightFielderFiles WITH DataPath
```

3 Call the **InitializeDataFiles** function to connect the RightFielder Object to its supporting data file.

```
CALL InitializeDataFiles RETURNING Result
IF Result <> ErrorNone Then
   CALL GetInitializeErrorString RETURNING ErrorString
    PRINT "Error: " & ErrorString
ENDIF
```

4 If information in your input doesn't match one of RightFielder Object's predefined data types, you can define a user pattern to catch other types, such as Social Security numbers, using regular expressions.

```
CALL SetUserPattern WITH ("SSN", "[0-9]{3}-[0-9]{2}-[0-
   9]{4}"
```

5 Call the parsing function Parse. **ParseFreeform (Deprecated)ParseFielded (Deprecated)**

```
CALL Parse with InputString
```

6 After calling the parsing function, retrieve the parsed data.

```
CALL GetAddress Returning AddressLine1
CALL GetAddress2 Returning AddressLine2
CALL GetAddress3 Returning AddressLine3
CALL GetCity Returning City
CALL GetState Returning State
CALL GetPostalCode Returning PostalCode
CALL GetCountry Returning Country
```

7 Some of the output functions can return multiple instances of the same data type. To return them separately, call the respective **Get...Next** function.

```
CALL GetFullName Returning FullName
APPEND FullName to FullNames Array
WHILE GetFullNameNext Returns True
    CALL GetFullName Returning FullName
    APPEND FullName to FullNames Array
END WHILE
```

Similar logic can be applied to the **GetCompany**, **GetDepartment**, **GetEmail**, **GetPhone**, **GetPhoneType**, **GetURL** and **GetUnrecognized**.

8 Retrieve the information that matched your user pattern.

```
CALL GetUserField WITH "SSN" Returning SSN
APPEND SSN to SSNS Array
WHILE CALLING GetUserFieldNext WITH "SSN" Returns True
    CALL GetUserField WITH "SSN" Returning SSN
    APPEND SSN to SSNS Array
END WHILE
```

Chapter 2
RightFielder Object Functions

2.1: RightFielder Object Functions

This is a list of the available functions accessible via the RightFielder Object programming interface.

2.2.1: Initialize the RightFielder Object

These functions initialize RightFielder Object and connect it to its data files.

2.2.2: Set RightFielder Object Options (Deprecated)

The functions in this section are deprecated and exist only to ensure backwards compatibility, a call to these functions do not effect processing, as the object ignores the calling option, and internally uses advanced data recognition to determine whether a specific datatype should be accepted or not.

The functions in this section are used to configure the available options for RightFielder Object. These methods have been deprecated, but exist for backwards compatibility.

2.2.3: Advanced Settings

The following functions configure the advanced features of RightFielder Object. These functions should be considered optional and for experienced users.

2.2.4: Parse Input Data

The following functions clear the previous results, parse the input strings, and populate the return values. ParseFreeform and ParseFielded are deprecated.

2.2.5: Retrieve Parsed Values

The following functions retrieve the information that has been extracted from the input string by the parse function.

2.2: RightFielder Object Function Reference

The function reference is broken up into sections based on the typical order of operations when using RightFielder Object.

2.2.1: Initialize the RightFielder Object

These functions initialize RightFielder Object and connect it to its data files.

SetPathToRightFielderFiles

This function accepts a string value containing the file path to the data files that support the RightFielder Object.

Remarks

The value passed to this function must contain the full file path to the following file: mdRightFielder.dat

Syntax

```
object->SetPathToRightFielderFiles(string);
```

C

```
mdRightFielderSetPathToRightFielderFiles(object, char*);
```

COM

```
object.PathToRightFielderFiles = string
```

SetLicenseString

The License String is a software key that unlocks the full functionality of the component. Without the License String, the object will not function.

VB Example

```
To set your License String . . .
RightFielder.SetLicenseString ("pass your License String here")
'Set the paths to the data files.
```

```
PathToRightFielderFiles = "C:\Program Files\Melissa
DATA\DQT\Data\"
'Initialize the data files.
If (RightFielder.InitializeDataFiles () <> 0) Then
       'If InitializeDataFiles returned anything other then 0
       'display the error
       MsgBox (addPtr.GetInitializeErrorString())
End if
```

Input Parameters

The SetLicenseString function has one parameter.

LicenseString	A required string value representing the software license key.

While it's possible to set the license string by passing it directly via this function, it is most often preferable to set the license string into an environment variable called MD_LICENSE.

Using an environment variable makes it much easier to update the license string without having to edit and re-compile the application.

When using an environment variable, it is not necessary to call the SetLicenseString function.

For more information on setting the environment variable, see page 2 of this guide.

Return Value

The SetLicenseString function returns a Boolean value of 0 (FALSE) or 1 (TRUE). The SetLicenseString function will return a FALSE Boolean value if the License String provided is incorrect or empty..

Syntax

```
bool = object->SetLicenseString(LicenseString);
```

C

```
int = mdRightFielderSetLicenseString(object, LicenseString);
```

COM

```
boolean = object.SetLicenseString(LicenseString)
```

InitializeDataFiles

Opens the required data files and prepares the parsing logic for use.

Remarks

If the InitializeDataFiles function returns a code other than ErrorNone or zero, you can call the **GetInitializeErrorString** function to display a string describing the error.

You must set the license string environment variable or call the **SetLicenseString** function before calling this function.

You must call the **SetPathToRightFielderFiles** function before calling this function.

Return Value

The **InitializeDataFiles** function returns an enumerated value of the type ProgramStatus:

Enum Value (Integer Value)	Initialize Error String
ErrorNone (0)	"No Error"
No error occurred.	
ErrorConfigFile (1)	"Could not open mdRightFielder.dat"
mdRightFielder.dat could not be found at the locating specified when calling the **SetPathToRightFielderFiles** function.	
LicenseExpired (2)	"License expired"
The license string has expired. Contact Melissa Data to purchase a new license.	
Unknown (4)	"Unknown Error."
An error has occurred other than the specific ones listed here.	

For languages that do not accept enumerated values, this function accepts the matching integer value from the table.

Syntax

```
StringValue = object->InitializeDataFiles();
```

C

```
StringValue = mdRightFielderInitializeDataFiles(object);
```

COM

```
StringValue = object.InitializeDataFiles()
```

GetInitializeErrorString

Returns a descriptive string to describe the error from the InitializeDataFiles method.

Remarks

The GetInitializeErrorString method returns a string describing the error caused when the InitializeDataFiles method cannot be called successfully.

The possible strings returned by this method are listed in the table under the **InitializeDataFiles** function above.

Syntax

```
StringValue = object->GetInitializeErrorString()
```

C

```
StringValue = mdRightFielderGetInitializeErrorString(object)
```

COM

```
StringValue = object.GetInitializeErrorString
```

GetBuildNumber

The GetBuildNumber method returns the current development release build number of the RightFielder Object.

Syntax

```
StringValue = object->GetBuildNumber()
```

C

```
StringValue = mdRightFielderGetBuildNumber(object)
```

COM

```
StringValue = object.GetBuildNumber
```

GetDatabaseDate

The GetDatabaseDate method returns a string value that represents the date of your RightFielder Object data files.

Remarks

This date can be used to verify that the files used by RightFielder Object are the most current

Syntax

```
StringValue = object->GetDatabaseDate()
```

C

```
StringValue = mdRightFielderGetDatabaseDate(object)
```

COM

```
StringValue = object.GetDatabaseDate
```

GetLicenseExpirationDate

Returns a string value containing the expiration date of the current license string.

Remarks

Call this method to determine when your current license will expire. After this date, RightFielder Object will no longer function.

Syntax

```
StringValue = object->GetLicenseExpirationDate()
```

C

```
StringValue = mdRightFielderGetLicenseExpirationDate(object)
```

COM

```
StringValue = object.GetLicenseExpirationDate
```

2.2.2: Set RightFielder Object Options (Deprecated)

The functions in this section are deprecated and exist only to ensure backwards compatibility, a call to these functions do not effect processing, as the object ignores the calling option, and internally uses advanced data recognition to determine whether a specific datatype should be accepted or not.

SetAcceptAddress (Deprecated)

This function is deprecated. RightFielder Object now contains datatype recognition logic that automatically determines when this information is set.

This function enables the identification and storing of address information.

Remarks

Instructs RightFielder Object to identify address information in the input string and populate the return values of the **GetAddress**, **GetAddress2** and **GetAddress3** functions.

By default, this functionality is enabled. Pass a False or zero value to this function to disable it.

Syntax

```
object->SetAcceptAddress(boolean)
```

C

```
StringValue = mdRightFielderSetAcceptAddress(object, int)
```

COM

```
object.AcceptAddress = boolean
```

SetAcceptCityStateZip (Deprecated)

This function is deprecated. RightFielder Object now contains datatype recognition logic that automatically determines when this information is set.

This function enables the identification and storing of last line information (city name, state or province names, and ZIP or Postal codes).

Remarks

Instructs RightFielder Object to identify last line information in the input string and populate the return values of the **GetLastLine**, **GetCity**, **GetState**, and **GetPostalCode** functions.

By default, this functionality is enabled. Pass a False or zero value to this function to disable it.

Syntax

```
object->SetAcceptCityStateZip(boolean)
```

C

```
mdRightFielderSetAcceptCityStateZip(object, int)
```

COM

```
object.AcceptCityStateZip = boolean
```

SetAcceptCompany (Deprecated)

This function is deprecated. RightFielder Object now contains datatype recognition logic that automatically determines when this information is set.

This function enables the identification and storing of company names.

Remarks

Instructs RightFielder Object to identify company names in the input string and populate the return value of the **GetCompany** function.

By default, this functionality is enabled. Pass a False or zero value to this function to disable it.

Syntax

```
object->SetAcceptCompany(boolean)
```

C

```
mdRightFielderSetAcceptCompany(object, int)
```

COM

```
object.AcceptCompany = boolean
```

SetAcceptCountry (Deprecated)

This function is deprecated. RightFielder Object now contains datatype recognition logic that automatically determines when this information is set.

This function enables the identification and storing of country names.

Remarks

Instructs RightFielder Object to identify a country name in the input string and populate the return value of the **GetCountry** function.

By default, this functionality is enabled. Pass a False or zero value to this function to disable it.

Syntax

```
object->SetAcceptCountry(boolean)
```

C

```
mdRightFielderSetAcceptCountry(object, int)
```

COM

```
object.AcceptCountry = boolean
```

SetAcceptDepartment (Deprecated)

This function is deprecated. RightFielder Object now contains datatype recognition logic that automatically determines when this information is set.

This function enables the identification and storing of department names.

Remarks

Instructs RightFielder Object to identify a department name in the input string and populate the return value of the **GetDepartment** function.

By default, this functionality is enabled. Pass a False or zero value to this function to disable it.

Syntax

```
object->SetAcceptDepartment(boolean)
```

C

```
mdRightFielderSetAcceptDepartment(object, int)
```

COM

```
object.AcceptDepartment = boolean
```

SetAcceptEmail (Deprecated)

This function is deprecated. RightFielder Object now contains datatype recognition logic that automatically determines when this information is set.

This function enables the identification and storing of email addresses.

Remarks

Instructs RightFielder Object to identify an email address in the input string and populate the return value of the **GetEmail** function.

By default, this functionality is enabled. Pass a False or zero value to this function to disable it.

Syntax

```
object->SetAcceptEmail(boolean)
```

C

```
mdRightFielderSetAcceptEmail(object, int)
```

COM

```
object.AcceptEmail = boolean
```

SetAcceptFullName (Deprecated)

This function is deprecated. RightFielder Object now contains datatype recognition logic that automatically determines when this information is set.

This function enables the identification and storing of personal names.

Remarks

Instructs RightFielder Object to identify a person's name in the input string and populate the return value of the **GetFullName** function.

By default, this functionality is enabled. Pass a False or zero value to this function to disable it.

Syntax

```
object->SetAcceptFullName(boolean)
```

C

```
mdRightFielderSetAcceptFullName(object, int)
```

COM

```
object.AcceptFullName = boolean
```

SetAcceptPhone (Deprecated)

This function is deprecated. RightFielder Object now contains datatype recognition logic that automatically determines when this information is set.

This function enables the identification and storing of phone numbers.

Remarks

Instructs RightFielder Object to identify a phone number in the input string and populate the return value of the **GetPhone** function.

By default, this functionality is enabled. Pass a False or zero value to this function to disable it.

Syntax

```
object->SetAcceptPhone(boolean)
```

C

```
mdRightFielderSetAcceptPhone(object, int)
```

COM

```
object.AcceptPhone = boolean
```

SetAcceptURL (Deprecated)

This function is deprecated. RightFielder Object now contains datatype recognition logic that automatically determines when this information is set.

This function enables the identification and storing of URLs.

Remarks

Instructs RightFielder Object to identify a URL in the input string and populate the return value of the **GetURL** function.

By default, this functionality is enabled. Pass a False or zero value to this function to disable it.

Syntax

```
object->SetAcceptURL(boolean)
```

C

```
mdRightFielderSetAcceptURL(object, int)
```

COM

```
object.AcceptURL = boolean
```

2.2.3: Advanced Settings

The following functions configure the advanced features of RightFielder Object. These functions should be considered optional and for experienced users.

SetUserPattern

This function defines a regular expression that enables RightFielder Object to recognize a custom data type.

Remarks

This function uses regular expressions to define a custom data type, such as a social security number, a date, or a foreign phone number.

Examples

[0-9]{3}-[0-9]{2}-[0-9]{4}	Social Security number
\$[0-9]*.[0-9][0-9]	Dollar amount with leading "$."
[0-9]{4}-[0-9]{2}-[0-9]{2}	Date in numeric format, such as 2009-12-27.
[A-Z][a-z][a-z] [0-9][0-9]*, [0-9]{4}	Date in text format, such as: Dec 27, 2009.

Input Parameters

- **Description** — You can define multiple user patterns, so each one needs a name to distinguish it from the others. It should be easy to identify because it will be used to retrieve the data parsed using this User Pattern. The description text is case sensitive, so you cannot enter "ssn" and use "SSN" to retrieve matching text later.
- **RegEx** — the regular expression used to identify the custom data type.

Syntax

```
object->SetUserPattern(stringValue Desc, stringvalue RegEx)
```

C

```
mdRightFielderSetUserPattern(object, StringValue Desc,
   StringValue RegEx)
```

COM

```
object.SetUserPattern(stringValue Desc, stringvalue RegEx)
```

SetDelimiter (Deprecated)

This function is deprecated. RightFielder Object now contains datatype recognition logic that automatically determines when this information is set.

This function declares which characters will be used as delimiters to define fields in the input string.

Remarks

By default, RightFielder Object uses tab characters as a delimiter when applying the ParseFielded parsing logic, but you can select a different character for RightFielder Object to use.

Input Parameters

This function accepts a single value of the enumerated type Delimiter.

0	Tab
1	Comma
2	Pipe - “|”
3	CRLF

For languages that do not accept enumerated values, this function accepts the matching integer value from the table.

Syntax

```
object->SetDelimiter(enum Delimiter)
```

C

```
mdRightFielderSetDelimiter(object, int Delimiter)
```

COM

```
object.SetDelimiter(enum Delimiter)
```

2.2.4: Parse Input Data

The following functions clear the previous results, parse the input strings, and populate the return values.

Parse

This function is the recommended processing function.

Input Parameters

InputData — A string value containing the data to be parsed.

Syntax

```
object->Parse(stringValue InputData)
```

C

```
mdRightFielderParse(object, StringValue InputData)
```

COM

```
object.Parse(stringValue InputData)
```

ParseFreeform (Deprecated)

This function is deprecated. A call to this function is redirected to the Parse function. See “Parse” on page 22.

This function parses a string of data that may or may not be consistently delimited.

Remarks

If you suspect that your data may not contain consistent field delimiters, such as commas or tabs, use this method.

If you are confident that your data is consistently delimited, the **ParseFielded (Deprecated)** function will probably be faster and more accurate.

Example

“John Q. Smith 1234 Main St Peoria, IL 12345”

Input Parameters

InputData — A string value containing the data to be parsed.

Syntax

```
object->ParseFreeform(stringValue InputData)
```

C

```
mdRightFielderParseFreeform(object, StringValue InputData)
```

COM

```
object.ParseFreeform(stringValue InputData)
```

ParseFielded (Deprecated)

This function is deprecated. A call to this function is redirected to the Parse function. See “Parse” on page 22.

This function parses a string of data that is consistently delimited.

Remarks

If you are confident that your data is consistently delimited, the ParseFielded method will probably be faster and more accurate that the **ParseFreeform (Deprecated)** function.

Example

“John Q. Smith, 1234 Main St, Peoria, IL 12345”

Note

If the ParseFielded function is used, the Get...Next functions can be used to retrieve multiple instances of certain data types. For example, if more than person’s name is found in the input string, the **GetFullName** function will return the first instance of a name. If the **GetFullNameNext** function returns True, a second full name was found. Call the **GetFullName** function again to retrieve the second value.

Input Parameters

InputData — A string value containing the data to be parsed.

Syntax

```
object->ParseFielded(stringValue InputData)
```

C

```
mdRightFielderParseFielded(object, StringValue InputData)
```

COM

```
object.ParseFielded(stringValue InputData)
```

2.2.5: Retrieve Parsed Values

The following functions retrieve the information that has been extracted from the input string by the parse function.

GetResults

This function returns a string value containing status and error codes for the current record. Multiple codes are separated by commas.

Remarks

The GetResults function may return one or more four-character strings, separated by commas, depending on the result generated by the current record.

The possible values are:

Code	Short Desc.	Long Desc.
RS01	Parse Successful	The parse was successful.
RS02	Foreign Address Detected	The detected country was one that is not supported by RightFielder.
RE01	Unrecognized Data Present	The return value of the GetUnrecognized function has been populated.
RE02	Incomplete Data	The input string did not contain a complete and verifiable address.
RE03	Demo Mode	Rightfielder is in demo mode. This is due to an invalid or expired license key, or expired database.

Syntax

```
StringValue = object->GetResults()
```

C

```
StringValue = mdRightFielderGetResults(object)
```

COM

```
StringValue = object.Results
```

GetAddress

This function returns the street address information from the input string.

Remarks

If this function does not return a value no address information was detected.

Syntax

```
StringValue = object->GetAddress()
```

C

```
StringValue mdRightFielderGetAddress(object)
```

COM

```
StringValue = object.Address
```

GetAddress2

This function returns the second line of street address information, if any, from the input string.

Remarks

If this function does not return a value, a second line of address information was not detected.

Syntax

```
StringValue = object->GetAddress2()
```

C

```
StringValue mdRightFielderGetAddress2(object)
```

COM

```
StringValue = object.Address2
```

GetAddress3

This function returns the third line of street address information, if any, from the input string.

Remarks

If this function does not return a value, a third line of address information was not detected.

Syntax

```
StringValue = object->GetAddress3()
```

C

```
StringValue mdRightFielderGetAddress3(object)
```

COM

```
StringValue = object.Address3
```

GetFullName

This function returns any personal names detected in the input string.

Remarks

This function will return the first personal name that was detected. Call the **GetFullNameNext** function. If it returns a True value, a second personal name was found. Call this function again to retrieve that name. If the **GetFullNameNext** function returns False, no more personal names were found.

Syntax

```
StringValue = object->GetFullName()
```

C

```
StringValue = mdRightFielderGetFullName(object)
```

COM

```
StringValue = object.FullName
```

GetFullNameNext

This function indicates if more personal names were found and, if true, updates the return value of the **GetFullName** function.

Remarks

If more than one personal name was found, this function returns a True value and populates the return value of the **GetFullName** function. If no more personal names can be retrieved, this function returns a False value.

If the **ParseFreeform (Deprecated)** function was used, this function will always return a False value.

Syntax

```
Boolean = object->GetFullNameNext()
```

C

```
Integer = mdRightFielderGetFullNameNext(object)
```

COM

```
Boolean = object.GetFullNameNext
```

GetCity

This function returns the city name, if one was found in the input string.

Remarks

To retrieve the city, state and postal code as a single line of text, use the **GetLastLine** function.

Syntax

```
StringValue = object->GetCity()
```

C

```
StringValue = mdRightFielderGetCity(object)
```

COM

```
StringValue = object.City
```

GetState

This function returns the state abbreviation, if one was found in the input string.

Remarks

To retrieve the city, state and postal code as a single line of text, use the **GetLastLine** function.

Syntax

```
StringValue = object->GetState()
```

C

```
StringValue = mdRightFielderGetState(object)
```

COM

```
StringValue = object.State
```

GetPostalCode

This function returns the ZIP Code™ or Postal Code, if any were found in the input string.

Remarks

This function will return a five-digit ZIP Code, a nine digit ZIP + 4®, or a Canadian Postal Code.

To retrieve the city, state and postal code as a single line of text, use the **GetLastLine** function.

Syntax

```
StringValue = object->GetPostalCode()
```

C

```
StringValue = mdRightFielderGetPostalCode(object)
```

COM

```
StringValue = object.PostalCode
```

GetLastLine

This function returns a formatted last line with city, state, and ZIP or Postal code.

Remarks

To retrieve the city, state and ZIP/Postal code as separate data, use the **GetCity**, **GetState**, and **GetPostalCode** functions.

Syntax

```
StringValue = object->GetLastLine()
```

C

```
StringValue = mdRightFielderGetLastLine(object)
```

COM

```
StringValue = object.LastLine
```

GetCountry

This function returns a country name or abbreviation, if one was detected in the input string.

Remarks

The GetCountry function will return country names and abbreviations such as “U.S.,” “United States,” “Canada,” or “CAN.”

RightFielder Object will also standardize the country name from U.S. and Canadian addresses. It may recognize the names of other nations but will not standardize their names.

Syntax

```
StringValue = object->GetCountry()
```

C

```
StringValue = mdRightFielderGetCountry(object)
```

COM

```
StringValue = object.Country
```

GetCompany

This function returns a company name or names, if any were detected in the input string.

Remarks

This function will return the first company name that was detected. Call the **GetCompanyNext** function. If it returns a True value, a second company name was found. Call this function again to retrieve that name. If the **GetCompanyNext** function returns False, no more company names were found.

Syntax

```
StringValue = object->GetCompany()
```

C

```
StringValue = mdRightFielderGetCompany(object)
```

COM

```
StringValue = object.Company
```

GetCompanyNext

This function indicates if more company names were found and, if true, updates the return value of the **GetCompany** function.

Remarks

If more than one company name was found, this function returns a True value and populates the return value of the **GetCompany** function. If no more company names can be retrieved, this function returns a False value.

If the **ParseFreeform (Deprecated)** function was used, this function will always return a False value.

Syntax

```
Boolean = object->GetCompanyNext()
```

C

```
Integer = mdRightFielderGetCompanyNext(object)
```

COM

```
Boolean = object.GetCompanyNext
```

GetDepartment

This function returns the name or names of departments within a company, if any were detected in the input string.

Remarks

This function returns the first department name that was detected. Call the **GetDepartmentNext** function. If it returns a True value, an additional department name was found. Call this function again to retrieve that name. If the **GetDepartmentNext** function returns False, no more department names were found.

Syntax

```
StringValue = object->GetDepartment()
```

C

```
StringValue = mdRightFielderGetDepartment(object)
```

COM

```
StringValue = object.Department
```

GetDepartmentNext

This function indicates if more department names were found and, if true, updates the return value of the **GetDepartment** function.

Remarks

If more than one department name was found, this function returns a True value and populates the return value of the **GetDepartment** function. If no more department names can be retrieved, this function returns a False value.

If the **ParseFreeform (Deprecated)** function was used, this function will always return a False value.

Syntax

```
Boolean = object->GetDepartmentNext()
```

C

```
Integer = mdRightFielderGetDepartmentNext(object)
```

COM

```
Boolean = object.GetDepartmentNext
```

GetEmail

This function returns email addresses, if any were detected in the input string.

Remarks

This function returns the first email address that was detected. Call the **GetEmailNext** function. If it returns a True value, an additional email address was found. Call this function again to retrieve that address. If the **GetEmailNext** function returns False, no more email addresses were found.

Syntax

```
StringValue = object->GetEmail()
```

C

```
StringValue = mdRightFielderGetEmail(object)
```

COM

```
StringValue = object.Email
```

GetEmailNext

This function indicates if more email addresses were found and, if true, updates the return value of the **GetEmail** function.

Remarks

If the **ParseFielded (Deprecated)** function was used to parse the input data and more than one email address was found, this function returns a True value and populates the return value of the **GetEmail** function. If no more email addresses can be retrieved, this function returns a False value.

If the **ParseFreeform (Deprecated)** function was used, this function will always return a False value.

Syntax

```
Boolean = object->GetEmailNext()
```

C

```
Integer = mdRightFielderGetEmailNext(object)
```

COM

```
Boolean = object.GetEmailNext
```

GetPhone

This function returns phone numbers, if any were detected in the input string.

Remarks

This function will return the first phone number that was detected. Call the **GetPhoneNext** function. If it returns a True value, an additional phone number was

found. Call this function again to retrieve that number. If the **GetPhoneNext** function returns False, no more phone numbers were found.

Syntax

```
StringValue = object->GetPhone()
```

C

```
StringValue = mdRightFielderGetPhone(object)
```

COM

```
StringValue = object.Phone
```

GetPhoneNext

This function indicates if more phone numbers were found and, if true, updates the return value of the **GetPhone** function.

Remarks

If more than one phone number was found, this function returns a True value and populates the return value of the **GetPhone** function. If no more phone numbers can be retrieved, this function returns a False value.

If the **ParseFreeform (Deprecated)** function was used, this function will always return a False value.

Syntax

```
Boolean = object->GetPhoneNext()
```

C

```
Integer = mdRightFielderGetPhoneNext(object)
```

COM

```
Boolean = object.GetPhoneNext
```

GetPhoneType

This function returns descriptive phone number tags, such as “Home” or “Cell,” if any were detected in the input string.

Remarks

If a phone number was tagged with a label such as “Home,” “Cell,” or “Work,” that label will be returned.

This function will return the first phone number label that was detected. Call the **GetPhoneTypeNext** function. If it returns a True value, an additional phone number label was found. Call this function again to retrieve that label. If the **GetPhoneTypeNext** function returns False, no more phone number labels were found.

Syntax

```
StringValue = object->GetPhoneType()
```

C

```
StringValue = mdRightFielderGetPhoneType(object)
```

COM

```
StringValue = object.PhoneType
```

GetPhoneTypeNext

This function indicates if more phone number labels were found and, if true, updates the return value of the **GetPhoneType** function.

Remarks

If more than one phone number label was found, this function returns a True value and populates the return value of the **GetPhoneType** function. If no more phone number labels can be retrieved, this function returns a False value.

If the **ParseFreeform (Deprecated)** function was used, this function will always return a False value.

Syntax

```
Boolean = object->GetPhoneTypeNext()
```

C

```
Integer = mdRightFielderGetPhoneTypeNext(object)
```

COM

```
Boolean = object.GetPhoneTypeNext
```

GetURL

This function returns uniform resource locators (URLs or web addresses), if any were detected in the input string.

Remarks

This function returns the first URL that was detected. Call the **GetURLNext** function. If it returns a True value, an additional URL was found. Call this function again to retrieve that URL. If the **GetURLNext** function returns False, no more URLs were found.

Syntax

```
StringValue = object->GetURL()
```

C

```
StringValue = mdRightFielderGetURL(object)
```

COM

```
StringValue = object.URL
```

GetURLNext

This function indicates if more URLs were found using the RightFielderParse method and, if true, updates the return value of the **GetURL** function.

Remarks

If more than one URL was found, this function returns a True value and populates the return value of the **GetURL** function. If no more URLs can be retrieved, this function returns a False value.

If the **ParseFreeform (Deprecated)** function was used, this function will always return a False value.

Syntax

```
Boolean = object->GetURLNext()
```

C

```
Integer = mdRightFielderGetURLNext(object)
```

COM

```
Boolean = object.GetURLNext
```

GetUnrecognized

This function returns any unmatched portions of the input string.

Remarks

This function returns the first string of unrecognized text that was detected. Call the **GetUnrecognizedNext** function. If it returns a True value, an additional string of unrecognized text was found. Call this function again to retrieve that data. If the **GetUnrecognizedNext** function returns False, no more unmatched text was found.

Syntax

```
StringValue = object->GetUnrecognized()
```

C

```
StringValue = mdRightFielderGetUnrecognized(object)
```

COM

```
StringValue = object.Unrecognized
```

GetUnrecognizedNext

This function indicates if more strings of unrecognized text were found and, if true, updates the return value of the **GetUnrecognized** function.

Remarks

If more than one string of unrecognized text was found, this function returns a True value and populates the return value of the **GetUnrecognized** function. If no more unrecognized strings of text can be retrieved, this function returns a False value.

If the **ParseFreeform (Deprecated)** function was used, this function will always return a False value.

Syntax

```
Boolean = object->GetUnrecognizedNext()
```

C

```
Integer = mdRightFielderGetUnrecognizedNext(object)
```

COM

```
Boolean = object.GetUnrecognizedNext
```

GetUserField

This function returns strings of text that match a named user pattern defined by the SetUserPattern function.

Remarks

This function accepts a string that matches a pattern name that was passed to the **SetUserPattern** function when creating a user pattern, and returns the first string of matching text that was detected. Call the **GetUserFieldNext** function . If it returns a True value, an additional string of matching text was found. Call this function again to retrieve that data. If the **GetUserFieldNext** function returns False, no more matching text was found.

Input Parameters

- **Descriptor** — This string value must exactly match the name given to a user pattern when it was created with the **SetUserPattern** function. This string value is case sensitive. “SSN” will not match “Ssn.”

Syntax

```
StringValue = object->GetUserField(StringValue Descriptor)
```

C

```
StringValue = mdRightFielderGetUserField(object, StringValue
   Descriptor)
```

COM

```
StringValue = object.UserField(StringValue Descriptor)
```

GetUserFieldNext

This function indicates if more strings of text matching the named user pattern were found and, if true, updates the return value of the **GetUserField** function for that pattern.

Remarks

This function accepts a string that matches the pattern name that was passed to the **SetUserPattern** function when creating a user pattern, and if more than one string of matching text for the named pattern was found, this function returns a True value and populates the return value of the **GetUserField** function for that pattern. If no more matching strings of text can be retrieved, this function returns a False value.

If the **ParseFreeform (Deprecated)** function was used, this function will always return a False value.

Input Parameters

- **Descriptor** — This string value must exactly match the name given to a user pattern when it was created with the **SetUserPattern** function. This string value is case sensitive. “SSN” will not match “Ssn.”

Syntax

```
StringValue = object->GetUserFieldNext(StringValue Descriptor)
```

C

```
StringValue = mdRightFielderGetUserFieldNext(object,
    StringValue Descriptor)
```

COM

```
StringValue = object.GetUserFieldNext(StringValue Descriptor)
```

Appendix

Modifying Settings for RightFielder Object

The data directory for Right Fielder Object includes a plain text file named mdRightFielder.cfg, which allows you to add to and override some of RightFielder Object's default token recognition. You can add entries to and remove entries from the installed tables of:

- [LeftToken] = List of words/phrases that commonly appear at the start of data (ie, name, company, title identifiers).
- [MiddleToken] = List of words/phrases that commonly appear in the middle of the data (ie, street address identifiers).
- [RightToken] = List of words/phrases that commonly appear in at the right end of the data (ie, city, state, zip, country identifiers).
- [LeftPattern] = Identifiable sequences of [LeftTokens].
- [MiddlePattern] = Identifiable sequences of [MiddleTokens].
- [RightPattern] = Identifiable sequences of [RightTokens].
- [PreProcessRegEx] = Regular expression search/replace that is performed on the raw input data before any other processing. This is often used to handle special anomolies in your data.
- [PostalCodeRegEx] = Regular expressions used to identify postal codes.
- [PhoneRegEx] = Regular expressions used to identify and extract phone numbers.
- [PhoneTypeToken] = List of words commonly used to identify phone numbers (Cell, Fax, etc)
- [EmailRegEx] = Regular expressions used to identify and extract e-mail addresses.
- [EmailTypeToken] = List of words commonly used to identify e-mail addresses (E-Mail, Email, etc)
- [URLRegEx] = Regular expressions used to identify and extract URLs.
- [URLTypeToken] = List of words commonly used to identify e-mail addresses (URL, website, etc)

Each section of the file begins with the name of the table enclosed in square brackets. To add an entry to the table, simply add a line to that section. If that entry duplicates one in the default table, mdRightFielder uses the entry in the configuration file, allowing you to override the original settings.

To remove an existing entry from the default table, begin the line with a dash or minus sign ("-").

An entry consists of a single line of text containing several values separated by commas.

For example, a line in the LeftToken section of the file must follow this format:

```
<token>, <tokenType>
```

Replacing each placeholder with a real value, the actual entry might look like this:

```
SMOKY THE BEAR,F
```

Unused values at the end of a line are optional. Empty values in the middle of a line must have the necessary comma.

Warning

This is an advanced topic, with descriptions and syntax for overriding the default tables are in the actual mdRightFielder.cfg. Please contact Melissa Data support with questions or advice before editing these tables, as any changes can significantly change the default behavior of the processor.