

Trillium Software System™ Programmer's Guide, Volume 1

includes:

API Overview Information
Calling all modules on Windows
Calling all modules on UNIX
The Universal Cleansing Adapter Function

Version 7.16

May 2017

**TRILLIUM
SOFTWARE™**

This manual, as well as the software described in it, are furnished under license and may be used only in accordance with the terms of such license. The content of this manual is furnished for informational purposes only, is subject to change without notice, and should not be construed as a commitment by Trillium Software. Trillium Software assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual.

The customer shall not disclose, copy, reproduce, distribute, or display any portion of the Trillium Software System or this manual in any form to any third person without the prior written consent of Trillium Software, nor allow third parties to do the same. The customer shall keep the Trillium Software System and all confidential information in the strictest confidence.

Trillium Software System Programmer's Guide, Volume 1 53017

Trillium Software, Inc. owns all rights in and to the marks "TRILLIUM SOFTWARE" and "TRILLIUM SOFTWARE SYSTEM," which marks are registered in various countries throughout the world (including, without limitation, the United States Patent and Trademark Office).

All other trademarks are the property of their respective owners.

© 2008-2017 Trillium Software, Inc.

Table of Contents

CHAPTER 1	Trillium Software System Applications Programming Interface (API) Overview	1-1
	API Overview	1-2
	Trillium Software System API Specifics	1-3
	Types of Calls	1-3
	TSS Universal Cleansing Adapter (UCA)	1-4
	TSS API Formats	1-5
	TSS-Supported Functions	1-6
	API Group Details	1-7
	API Group 1	1-7
	API Group 2	1-11
	API Group 3	1-13
	API Group 4	1-14
	Single vs. Multi-Threaded Applications.....	1-15
	Naming Conventions	1-16
	Linking the APIs	1-17
	Typical API Flow	1-18
	Calling from Java.....	1-19
CHAPTER 2	Universal Cleansing Adapter (UCA)	2-1
	Universal Cleansing Adapter Functions	2-3
	UCA Process Diagram.....	2-4
	Configuring the parmfile.ini File	2-5
	parmfile.ini Parameters.....	2-7
	Location of Libraries	2-11
	Class Files.....	2-11
	Setting up the Sample UCA program	2-12
	API Project Files.....	2-12
	Compiling and Executing the TestEZCallable.java Program	2-12
	UCA Methods Calling Order	2-14
	UCA Calling Methods Details	2-16
	List of UCA-Derived Fields.....	2-28

CHAPTER 3	Trillium Software System API Calls on Windows	3-1
	API Calls from C/C++ on Windows.....	3-2
	API Calls from C# on Windows	3-2
	Call types for the 'Action' Parameter	3-3
	Calling the Converter API from C#	3-4
	Calling the Business Data Parser API from C#	3-5
	Calling the Create Window Key API from C#	3-6
	Calling the Matcher API from C#	3-7
	API Calls from VB .NET.....	3-28
	Call types for the 'Action' Parameter	3-29
	Calling the Converter API from VB .NET	3-30
	Calling the Business Data Parser API from VB .NET	3-31
	Calling the Create Window Key API from VB .NET	3-32
	Calling the Matcher API from VB .NET	3-33
	API Calls from Visual Basic.....	3-52
	VB call types for the 'Action' parameter.....	3-52
	Calling the Converter API from VB.....	3-53
	Calling the Business Data Parser API from VB	3-55
	Calling the Create Window Key API from VB	3-56
	Calling the Matcher API from VB	3-57
CHAPTER 4	Trillium Software System API Calls on UNIX	4-1
	API Calls from C on UNIX.....	4-2
	'C' Call Types for the 'Action' Parameter	4-2
	Calling the Converter API from C.....	4-4
	Calling the Business Data Parser API from C	4-5
	Calling the Create Window Key API from C	4-7
	Calling the Matcher API from C	4-8
	Calling the RDI Module from C/C++	4-29
	Calling the XML APIs from C.....	4-32
	Other XML Functions	4-35
	API Calls from C++ on UNIX	4-39
	API Calls from Java on UNIX	4-39
	Calling the Converter API from Java	4-40
	Calling the Business Data Parser API from Java	4-42
	Calling the Create Window Key API from Java	4-44
	Calling the Matcher API from Java	4-45
	Reference Matching Mode.....	4-46

Window Matching.....4-48

Index I-1

CHAPTER 1

Trillium Software System Applications Programming Interface (API) Overview

This document set is designed to help developers create programs that execute a call to a Trillium Software System™ (TSS) function. The set is organized as outlined below.

Volume 1 contains the following information:

- Chapter 1: An overview of the TSS API's, including the Universal Cleansing Adapter (UCA)
- Chapter 2: A detailed description of the UCA.
- Chapter 3: For Windows operating systems, procedures for calling TSS modules that are not incorporated into the UCA.
- Chapter 4: For Unix operating systems, procedures for calling TSS modules that are not incorporated into the UCA.

Volume 2 contains the following information:

- Chapter 1: Introduction to the document.
- Chapter 2: The procedures for calling all the TSS modules from the IBM Mainframe.

- Chapter 3: Error codes and other data that is output from all of the TSS modules.

To get started, review the contents of each volume, find the platform you use, and then locate the programming language you use.

For additional assistance, call or e-mail Trillium Software Technical Support at **techsupport@trilliumsoft.com**.

① *Trillium Software recommends all development for name and address processing be performed by leveraging the UCA rather than through integration with the low-level APIs.*

API Overview

The Trillium Software System® (TSS) API's give customers the ability to call a TSS module from a real-time application. When these modules are called, the customer program takes on the role of the batch driver. Essentially, TSS API's are a set of building blocks for programmers; the API's meet the need to transparently query, analyze, standardize and match data in real-time.

The advantage of calling a TSS module (versus using it in batch) is that it enables TSS technology to be embedded in real-time applications such as a call center or Web-based information system. In this type of application, data can be entered into a Web form or some other type of input form. When the data is submitted, it is cleansed, standardized, and matched, allowing for rapid changes in data presentation to be applied. This reduces the need for future data handling.

Enriched and matched data is then returned to the calling application, in some cases allowing the user to make choices about what data should move onto the next process. This type of intelligent processing ensures that when data is presented to the database, it is more robust than it would have been otherwise. For many businesses, this real-time processing and decision making is essential when dealing with customer information.

① *Paths set within a parameter file cannot exceed 80 characters.*

Trillium Software System API Specifics

The TSS functions are a library of routines that can be called from any language that can make an external call to a C function.

This table shows some of the common platforms and languages that are supported:

	UNIX	IBM Mainframe	Windows	Sample Program for Language
JAVA	X		X	X
C/C++	X	X	X	
C#			X	X
COBOL		X		X
VB			X	
VB .NET			X	X
RPG IV				X

Types of Calls

A call to a Trillium Software System® module consists of several individual steps that are performed by the calling program. Without a driver to handle the input and output functions, the calling program must take on this responsibility.

Some modules (the Matcher in particular) have many individual calls to separate functions that must be performed in a particular order.

The following sections of this chapter detail the various types of calls that the TSS API's require.

TSS Universal Cleansing Adapter (UCA)

The Universal Cleansing Adapter (UCA) combines several TSS data cleansing modules, in order to simplify integration of the API's into the calling application.

This single API approach simplifies the task of pushing data through the modules because the programmer does not need to know the internal workings of each TSS function in order to successfully implement them. This interface also allows access to the cleansed data by field name (through an API function called **getField**), further simplifying the data processing.

The data cleansing modules available in the UCA API are:

- Global Data Router
- Customer Data Parser
- All Postal and Census Geocoders
- Data Reconstructor
- Window Key Generator
- Investigator

The UCA is available for the following operating systems **only**, and is the API to use for all applications where it is supported.

- Solaris
- Linux
- AIX
- Windows

The UCA can be called from the following languages

- C++
- C#
- VB.NET
- Java

See *Chapter 2, "Universal Cleansing Adapter (UCA),"* for a description of the UCA and how to use it.

For applications where the UCA is unavailable, please read the remainder of this introduction to familiarize yourself with the individual API's and their operation.

① *If you are creating an application that uses multi-threading of the transactions, see "Single vs. Multi-Threaded Applications" on page 1-15.*

TSS API Formats

TSS API's can be categorized into two groups:

- Single Multi-Action API modules
- Multi Single-Action API modules

A **single multi-action API module** is one that uses the same API calling structure to perform multiple operations. Some modules that fall into this category are the Converter, Parser, Geocoder(s) and the Window Key Generator.

All of these modules have one calling structure with a minimum of three operations: open, process and close. "O"pen initializes the module, "P"rocess performs operations on the data, and "C"lose cleans up after the module and writes any statistics.

The following examples show Converter calls to perform the three operations.

Open call: `cf_coni("O", &control, parmfile, parmecho, inarea, outarea, &retcode);`
Process call: `cf_coni("P", &control, parmfile, parmecho, inarea, outarea, &retcode);`
Close call: `cf_coni("C", &control, parmfile, parmecho, inarea, outarea, &retcode);`

The example above illustrates that for the three calls to the Converter, the parameters are consistent. The values of each parameter, such as the action parameter, change according to the call being executed.

A **multi single-action API** is one that uses different API parameter structures for each of the various operations. In this case, each API call is unique and performs only one operation.

The Matcher and XML Converters fall into this category. For example, if a

calling application wants to initialize the Matcher, process the parameters, and close the Matcher, it must contain the following calls:

```
Initialize_Matcher(&retcode, statfile, linkfile, &reclen, control);  
Process_Parameters(&retcode, parmfile, parmtype, NULL, NULL, control);  
End_Matcher(&retcode, control);
```

 *Paths set within a parameter file must be 80 characters or less.*

TSS-Supported Functions

The following section outlines each TSS function that is available as an API as well as some specific details pertaining to each function. The following TSS functions are available as an API for use within your online application:

- Global Data Router
- Investigator
- Converter
- Customer Data Parser
- Business Data Parser
- All Postal and Census Geocoders
- Window Key Generator
- Matcher
- Data Reconstructor
- XML converters

Please note: As explained previously, when calls are made to the TSS API's, the customer's calling program takes on the role of the TSS batch driver; functions performed by the TSS batch driver are not available.

To determine which batch functions do not exist in the API, review the equivalent batch driver parameter file. If the function exists in the parameter file, it is not available to use as a function in the TSS API module.

API Group Details

The following sections discuss the various API's and provide preliminary information for the developer. These sections should be reviewed before development of the calling application.

API Group 1

The following five TSS functions use the same calling architecture:

- Global Data Router
- Investigator
- Converter
- Window Key Generator
- Data Reconstructor

Each of the functions listed above are single multi-action API's. They require DDL's and a parameter file.

The suggested method for making calls is to make one **open call** to the function and process all transactions in a loop.

After all transactions are processed, a single **close call** is performed, which releases all resources allocated during the open call and produces any applicable statistics.

Global Data Router API

Calls to the Router API are optional, depending on the application. This function is used to identify transactions by country of origin and assign a country identifier to the record. The record can then be directed to the country-specific Parser and Geocoder.

Note that there are two Routers available from Trillium Software:

- The first is designed only for use with **single-byte data**, requiring data to be delivered to it in IBM code page 819.
- The second is designed to work with **both single byte and double byte data**, and is delivered as part of the Series 7 Unicode product. This version is known as the **ucRouter** and will accept data in multiple formats, one of which is Unicode. It can translate data from Unicode format into various other code pages for further processing with other TSS functions, as well as identify a record's country of origin.

The Router API requires a customized **rules file** which must be defined within the standard parameter file. In addition, input and output DDL's are also required to describe the shape of the input record.

 *For additional details regarding Router control rules, see the Global Data Router chapter of the **Batch User's Guide**.*

Investigator API

Calls to the Investigator API are an optional function, used for incoming field analysis on a record-by-record basis. The Investigator does not perform any reformatting or recoding of data.

If this functionality is not a requirement of the calling application, then calls to the Investigator may be omitted.

- ① *For a detailed listing of Investigator functionality, please see the Investigator chapter of the **Batch User's Guide**.*

Converter API

Calls to the Converter API are an optional function within a calling application. If the incoming data is already formatted properly for the Parser API, and there is no requirement to reformat, recode, or otherwise re-engineer the data, then calls to the Converter API may be omitted.

The Converter API uses both an input and output DDL to define the data's shape. Since the input and output DDLs are separate, their layouts may differ.

- ① *For a detailed listing of Converter functionality, please see the Converter chapter of the **Batch User's Guide**.*

Window Key Generator API

Calls to the Window Key Generator API are made to construct a composite key from elements of the input record, in preparation for calls to the Matcher API. This window key is used during a database query to select a candidate group of records that will be sent to the Matcher. The Window Key Generator uses a **customized rules file** to determine how to create these keys.

- ① *For details of Window Key functionality, please see the Window Key Generator chapter of the **Batch User's Guide**.*

Data Reconstructor API

Calls to the Data Reconstructor API are an optional function, for conditionally reformatting or reconstructing the layout of the data returned from TSS functions. It uses its own special **scripting language**. It also uses a rules-based parameter file, allowing the user to select nearly any combination of data for output.

If there is no requirement for this functionality, then calls to the Data Reconstructor API may be omitted.

- ① *For details of Data Reconstructor functionality, please see the Data Reconstructor chapter of the **Batch User's Guide**.*

API Group 2

The following functions use the same calling architecture:

- Customer Data Parser (CDP)
- Business Data Parser (BDP)

Both Parser functions are single multi-action API's, which require parameter files, but do not use DDL's.

The suggested method for making calls to both Parser API's is to make one **open** call to the function, **process** all transactions in a loop, then perform a single **close** call to release all of the resources and produce any applicable statistics.

Customer Data Parser (CDP) API

The CDP is used to identify elements within a record as either name or address data in preparation for geocoding and matching.

The CDP does not use DDL's to define the layout of the incoming data. The input buffer (which is passed to the CDP) must be populated by the calling program and adhere to the following requirements:

- Input lines must contain only name and address data.
- A maximum of ten (10) lines of up to 100 bytes each can be provided.
- The overall length of Parser input cannot exceed 1000 bytes.

The input layout is defined in LINE1-10 parameters within the **pfparser.par** file. These line definitions must represent the shape of the input data.

The Parser prefers the input data to be in a particular order. The name lines should come first, followed by the street lines, and then the geography lines.

It is best (but not required) to have all of the geography information (such as city, state and postal code) on the same input line to aid the Parser in identifying the components.

The return buffer contains 9836 bytes of data known as the **PREPOS**. This return buffer contains the input data as standardized and parsed elements.

Since functions performed by the CDP batch driver are not available within the API, any of the parameters available within the batch driver parameter file (such as **JOIN_LINES** and **CHANGE_DDNAME**) are not available. If needed, the customer's calling program must perform these services.

① *For details of Parser parameters and Parser functionality, please see the Customer Data Parser chapter of the **Batch User's Guide**.*

Business Data Parser (BDP) API

The BDP identifies **non-name and address**, mission-critical business data which is in need of standardization. The process is business rule-driven so that users can customize data identification according to specific requirements.

The BDP does not use DDL's to define the layout of incoming data. The input buffer must be populated by the calling program and adhere to the following requirements:

- Input may only contain non-name and address data.
- One line of up to 1000 bytes can be processed.

The data is defined within the **pfbparse.par** file as one miscellaneous (defined line type is "?") input line, containing up to 1000 characters.

The BDP return buffer contains 13,000 bytes of data known as the **BPREPOS**, which contains all of the non-name and address input data as parsed elements according to the user-defined definitions and patterns.

The **pfbparser.par** file contains references to the location of the business-specific and client-specific word and pattern tables.

① *For details of Parser parameters and Parser functionality, please see the BD Parser chapter of the **Batch User's Guide**.*

API Group 3

The following functions use the same calling architecture:

- Postal Geocoders (AU, CA, DE, HK, IT, TG, US, & UK)
- Census Geocoders (US Interpolated Rooftop, US Centroid Census)

All of the Geocoder functions listed above are single multi-action API's. They do not use parameter files or DDL's. The input and output areas are defined within the calling structure to each function. Input data to all of the geocoding functions must have been previously parsed.

US Postal Geocoding

For US Postal Geocoding, the input area passed to the function must contain the geocoder input area located in the PREPOS, returned from the CDP. This data is captured from the PREPOS and populated as input to the US Postal Geocoder. There are two output areas returned from the process call.

US Census Geocoding

For US Census Geocoding, the input data must first have been processed through the US Postal Geocoder. The output from the US Postal Geocoder must be passed as input to the US Census Geocoder. There are two output areas returned from the process call.

Other Geocoders

For all other country geocoders, the input area must first be created by running an additional routine, which will reformat elements from the PREPOS as input to the country-specific Geocoder.

The suggested method for making calls to all Geocoders is to make one **open** call to the function and **process** all transactions in a loop until the last transaction is reached. After all processing has completed, perform a single **close** call, which releases all resources and produces any applicable statistics.

API Group 4

The following functions use the same calling architecture:

- Name and Address Matcher API
- XML Converter API

Both modules are both single multi-action API's.

❗ *Unlike all other functions, each of these API's uses different functions to perform various operations.*

The suggested method for making calls to either module is to perform a single **open** and spawn multiple processes as needed. Close calls should be invoked to release resources or, when required, write statistics.

Single vs. Multi-Threaded Applications

Upon the **open** call for any TSS function, a control block (or “handle”) is created and returned to the calling program. This handle contains all of the information that was processed during the open, such as location of parameter information and the external resources, such as postal directories.

- ① *This handle MUST be preserved by the calling program and passed to all subsequent calls to the particular function.*

A **thread** is an encapsulation of the flow of control in a program. A single-threaded program executes one path through the application code at a time. For single-threaded applications, the handle (created in the open call) is maintained by the calling program and passed to each process call and then onto the close. Within this thread, the control area remains the same.

If users wish to multi-thread a process, the open is performed once and then the process call spawns multiple threads. The handle is passed to each process and it is a global variable used by every call related to that function. However, all input and output areas must be unique to *each thread*, therefore becoming local variables.

If input and output areas are not made local variables for each thread, one thread will be using the area of another. *Variables which are static and will not change per request may be kept global to the implementation.*

Naming Conventions

The TSS functions have different names than their batch driver equivalent executables. The Converter executable in the batch process is **cfcondrv**, but the Converter API can be a number of different names, depending on the calling program language.

For example, for a "C" call, the name is **cf_coni**, but for COBOL, it is **CF#CONI**. These names represent an entry point in the library file used by the calling program. This list shows the different entry names for the Converter:

C/C++	cf_coni
COBOL	CF#CONI
RPG	cf_coni
VB	_cf_coni@28

A detailed listing of entry point names for each API can be found in the header files, located in the **tril7vnn/include** directory of the server installation, where *nn* is the version number.

Linking the APIs

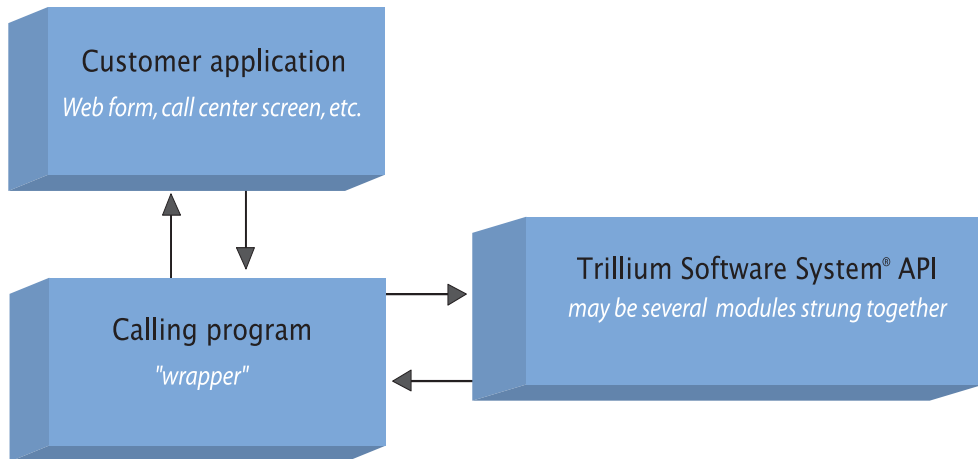
There are two ways to link in the API's: **dynamically** and **statically**. This table shows the types of linking that are supported on the different platforms, and the locations of the libraries.

	Dynamic	Static
IBM Mainframe	CALLABLE .LOADLIB	n/a
UNIX	./bin	./lib
Windows *	.\cdll .\stddl	.\lib

* For Windows, there are two sets of libraries. The .\cdll libraries are for languages like C/C++, while .\stddl libraries are for languages like Visual Basic.

Typical API Flow

A simple example of an API implementation using the Trillium Software System is represented by the diagram below:



There may be many layers of calling programs before the wrapper program that ultimately calls the TSS module. In addition, the wrapper program may be part of the application program.

The **wrapper program** handles specific tasks, like performing the initialization and passing the required parameters to the module. When the application has data to process by TSS, the wrapper program makes the "process" call to the TSS module, passes data to the module, and then retrieves the answer and passes the data back to the application (assuming that returning the data to the application is what is desired).

In this case, the wrapper's job is to be the "traffic cop" in the data flow.

Calling from Java

Calling from Java is similar to calling other C modules from other languages, only easier. Through the use of JNI (Java Native Interface), links to TSS C code have been accomplished. TSS functionality may be incorporated into an EJB architecture without the need to resolve native dependencies, as TSS API's are pre-packaged with these dependencies resolved.

The platform-specific dependency resulting from the use of JNI does not require compiling or porting as it has already been compiled and ported for you. The DLLs or Shared Libraries are provided as part of the Java API's.

Migrating Java server code from machine to machine requires only that an environment variable be set in order to resolve the locations of DLLs or Shared Libraries within a customer's particular system.

TSS modules can be called either **natively** or **over RMI**. Some advantages to implementing calls to modules via RMI include the capability of running the Trillium Software server on a different JVM than the application server. Another advantage of RMI is that the server and customer's Web service can run on two different machines or platforms, allowing the TSS server to use its own resources versus those of the web service.

Universal Cleansing Adapter (UCA)

The Universal Cleansing Adapter (UCA) is a single API function that can invoke the following modules.

- Global Data Router
- Customer Data Parser
- All Postal and Census Geocoders
- Data Reconstructor
- Investigator
- Window Key Generator

The modules listed above have independent API's, but this documentation set provides API information only for modules that cannot be called from the UCA.

The methods for calling the UCA, as well as detail on configuring the required resources, are provided in this chapter. In addition, instructions on using the sample program to call the UCA that is included with the Trillium Software System Server installation are in this chapter.

The UCA is available for the following operating systems:

- Solaris
- AIX
- Linux
- Windows operating systems

Each platform comes with its own set of shared objects (.so) or dynamic link libraries (.dll) files. These files are located in the ./bin directory of the Trillium Software System Server installation.

The UCA can be called from the following programming languages:

- Java
- C++
- VB.NET
- C#

Universal Cleansing Adapter Functions

The Universal Cleansing Adapter (UCA) contains the following Trillium Software System functions:

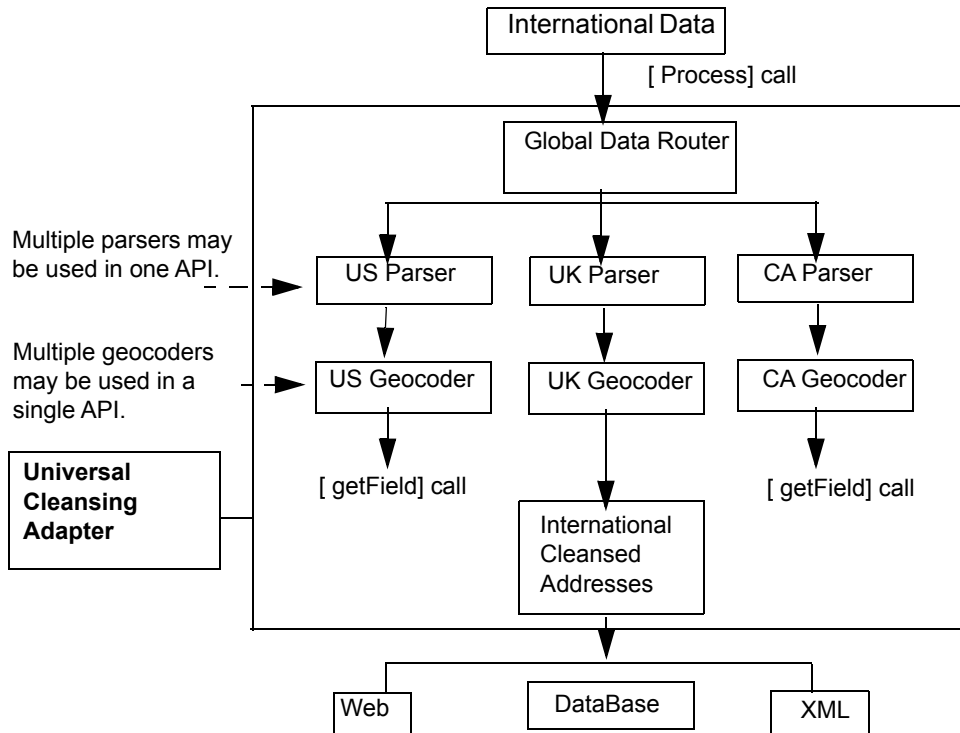
- Global Data Router (enabled if *pfrouter.par* is supplied)
- Customer Data Parser (one per country enabled)
- Data Reconstructor (enabled if *pfrecons.par* is supplied)
- Investigator
- Window Key generator
- Field retrieval function (for retrieving results)
- Postal Geocoder (one per country enabled, if available)
- US Census Geocoder

The Global Data Router is a unique part of the UCA process. It uses sophisticated algorithms and tables to determine the country of origin.

Each record is evaluated according to weighted criteria, and assigned the country which has the most weight. The country code for each transaction is then evaluated and directed to the appropriate country Parser and Postal Geocoder if available. This process is controlled and enabled through the UCA parameter file discussed later in this chapter.

UCA Process Diagram

This figure shows the process flow of data through the Router, Parsers, Geocoders, and the Universal Cleansing Adapter.



1. The Router parameter **COUNTRY_LIST** controls the list of countries that work with the UCA. This parameter takes precedence over the DefaultParser parameter provided in the UCA file (*parmfile.ini*).
2. After data is processed through the **process** method, a process handle is received, which is unique to that transaction record.
3. All of the output for that transaction can be obtained by calling **getField()** with that transaction's process handle.
4. When that transaction is completed, and data is no longer required, you should call **release()** with that transaction's process handle.

Configuring the *parmfile.ini* File

The file *parmfile.ini* is located in the samples folder of the Trillium Software System server installation. This file controls all functionality of the UCA. Inclusion of parameters in this file make individual modules available for use by the calling application, define resources to these modules, and control specific functions within each.

The file is structured in segments. Each segment defines an individual country or in some cases a specific module, if the module is not country-specific.

In your sample API project, change to the `\parms` folder (under `\uca_proj`) and open ***parmfile.ini***. Adjust the location of the files defined to the location where these files are located on your server.

The example below enables the following modules:

- Global Data Router
- Customer Data Parser (for US data)
- US Postal Geocoder
- Customer Data Parser (for Canadian data)
- CA Postal Geocoder
- Customer Data Parser (for UK data)
- UK Postal Geocoder
- Customer Data Parser (for Brazilian data)
- BR Postal Geocoder

Example *parmfile.ini* file

**nn* = Trillium Software System version number

```
showSplashScreen=yes
MaxNames=3
RouterParmFile=c:\tril7vnn\project\parms\pfrouter.par
RouterEchoFile= c:\tril7vnn\project\data\routerEcho
country_list=us,ca,uk
ReconsParmFile= c:\tril7vnn\project\parms\recons.txt
ReconsErrorFile= c:\tril7vnn\project\data\reconserror.txt
*****
```

2-6 Configuring the parmfile.ini File

```
CandidateCode1Routine=3
CandidateCode2Routine=4
CandidateCode3Routine=5
CandidateCode4Routine=6
CandidateCode5Routine=7
CandidateCode6Routine=8
CandidateCode7Routine=9
CandidateCode8Routine=1
////////////////////////////////////
USParserParmFile= c:\tril7vnn\project\parms\pfparsers.par
USPostalPath=c:\tril7vnn\tables
USCensusPath=c:\tril7vnn\tables
USFormFileName= c:\tril7vnn\project\data\usform.txt
USListName=Trillium List
USProcessorName=Trillium_Processor
USPostalDisplayBaseData=N
USCensusDisplayBaseData=N
////////////////////////////////////
CAParserParmFile= c:\tril7vnn\project\parms\pfparsers.par
CAPostalPath=c:\tril7vnn\tables
CAFormFileName= c:\tril7vnn\project\data\caform.txt
CAServiceBureauName=Canadian_Service_Bureau
CACustomerName=Canadian_Customer_Name
CACustomerAddress1=Canadian_Customer_Addr1
CACustomerAddress2=Canadian_Customer_Addr2
CACustomerAddress3=Canadian_Customer_Addr3
CACustomerAddress4=Canadian_Customer_Addr4
CAPostalDisplayBaseData=N
////////////////////////////////////
UKParserParmFile= c:\tril7vnn\project\parms\pfparsers.par
UKPostalPath=c:\tril7vnn\tables
UKFormFileName= c:\tril7vnn\project\data\ukform.txt
UKListName=Trillium
UKProcessorName=Trillium_Processor
UKPostalDisplayBaseData=N
////////////////////////////////////
BRParserParmFile= c:\tril7vnn\project\parms\pfparsers.par
BRPostalPath=c:\tril7vnn\tables
BRFormFileName= c:\tril7vnn\project\data\brform.txt
BRServiceBureauName=BR_Service_Bureau
BRCustomerAddress1=BR_Customer_Addr1
BRCustomerAddress2=BR_Customer_Addr2
BRCustomerAddress3=BR_Customer_Addr3
BRCustomerAddress4=BR_Customer_Addr4
BRPostalDisplayBaseData=N
BRLevelMatch=1,0,1,1,0,1,1,1,93,90,95,95,5,0,0,0
```

❗ *The **country_list** parameter in the Router parameter file takes precedence over the **country_list** parameter in parmfile.ini, so*

ensure sure these two entries are same.

*The directory paths in all parameter files (including parmfile.ini) need to be **absolute. No relative paths allowed.***

parmfile.ini Parameters

This table gives descriptions for the *parmfile.ini* parameters.

Note that **XX**=country code. All parameters in this table apply to all countries, unless explicitly stated otherwise.

① *Paths set within a parameter file cannot exceed 80 characters.*

Table 2.1 parmfile.ini Parameters

Parameter	Description
XXParserParmFile	Specifies the directory location and name of the Parser parameter file: CAParserParmfile=c:/tril7vnn/project/parms/pfparser.par
XXPostalPath	Specifies the directory location of Geocoder tables: UKPostalPath=c:/tril7vnn/tables
XXFormFileName	Specifies the directory location and name of the Form file: BRFormFileName=c:/tril7vnn/project/data/brform.txt
XXProcessorName	Specifies the name of the Processor: USProcessorName=Trillium_Processor
XXServiceBureauName	Specifies the name of the Service Bureau: NLServiceBureauName=NL_Service_Bureau
XXCustomerName	Specifies the name of the Customer: MXCustomerName=MX_Customer_Name
XXCustomerAddress1 XXCustomerAddress2 XXCustomerAddress3 XXCustomerAddress4	Specifies the address of the Customer: POCustomerAddress1=PO_Customer_Addr1 POCustomerAddress2=PO_Customer_Addr2 POCustomerAddress3=PO_Customer_Addr3 POCustomerAddress4=PO_Customer_Addr4

Table 2.1 parmfile.ini Parameters (Continued)

Parameter	Description
XXPostalDisplayBaseData	String (Y or N) or numeric value that specifies the Display BaseData values for a specified country: <code>DEPostalDisplayBaseData=N</code> or <code>SGPostalDisplayBaseData=5</code>
XXLevelMatch	Set of numeric values that specifies the criteria for matching; for example: <code>FRLevelMatch=1,1,1,1,1,1,1,1,92,95,100,100,6,0</code> See the table "Country-Specific TG Geocoder Parameters" in the TG Geocoder chapter in the <i>Geocoders</i> manual.
XXMaxNames	Numeric value (1—10) that determines how many names the Parser can write to the output record from one input record: <code>MaxNames=3</code> Applies to data quality connectors for Ascential DataStage and Informatica PowerCenter only.
CandidateCodeXRoutine (x=1-10)	Numeric value (0—9) that indicates which Candidate Code routine to use during parsing: <code>CandidateCode1Routine=3</code>
RouterParmFile	Specifies the directory location and name of the Router parameter file: <code>RouterParmFile=/myserver/tril7vnn/UCA_proj/parms/pfrouter.par</code>
RouterEchoFile	Specifies directory location and name of the Router echo (.log) file used for debugging: <code>RouterEchoFile=/myserver/tril7vnn/UCA_proj/data/router.log</code>
ReconsParmFile	Specifies the directory location and name of the Data Reconstructor parameter file: <code>ReconsParmFile=/myserver/tril7vnn/UCA_proj/parms/pfrecons.par</code>
ReconsEchoFile	Specifies directory location and name of the Data Reconstructor echo (.log) file used for debugging: <code>ReconsEchoFile=/myserver/tril7vnn/UCA_proj/data/recons.log</code>

Table 2.1 parmfile.ini Parameters (Continued)

Parameter	Description
WinkeyParmFile	<p>Specifies the directory location and name of the Window Key Generator parameter file:</p> <p><code>WinkeyParmFile=/myserver/tril7vnn/UCA_proj/parms/winkey.par</code></p> <p>Applies to the Trillium Software Director and the data quality connectors for Siebel and SAP only.</p>
Country-specific parameters	
XXDeliveryType	<p>Numeric value (0–2) that indicates the Delivery address format: <code>CZDeliveryType=1</code></p> <p>Applies to TG Geocoder countries: BE, BR, ES, FR, IT, MX, MY, PO, ES, NL, SG, and SW only.</p>
XXRulesFileName	<p>Specifies the name of the Rules file, which contains rules used to modify input data that enables the TG Geocoder to match street information to the postal directory:</p> <p><code>SGRulesFileName=SG_Rules_FileName</code></p> <p>Applies to TG Geocoder countries: BE, BR, ES, FR, IT, MX, MY, PO, ES, NL, SG, and SW only.</p>
XXListName	<p>Specifies the name of the List file (US or UK only):</p> <p><code>USListName=Trillium List (US)</code> <code>UKListName=Trillium (UK)</code></p>
US only	
USCensusPath	<p>Path that specifies the directory location of the US Census Geocoder tables:</p> <p><code>USCensusPath=c:/tril7vnn/tables/</code></p>
USHouseHoldPath	<p>Path that specifies the directory location of the US Interpolated Rooftop Census Geocoder tables:</p> <p><code>USHouseHoldPath=/tril14/tril_tables/</code></p>

Table 2.1 parmfile.ini Parameters (Continued)

Parameter	Description
USCensusDisplay BaseData	<p>String (Y or N, A, B, or M) or numeric (2–9) value that defines how and when the Centroid Census Geocoder applies a spelling algorithm to match street names to the postal directory:</p> <p><code>USCensusDisplayBaseData=N</code></p> <p>Centroid Census Geocoder DISPLAY_BASEDATA Values are listed in Volume II.</p>
USHouseHoldDisplay BaseData	<p>String (N) or numeric (4 or 5) value that defines how and when the Rooftop Census Geocoder applies a spelling algorithm to match street names to the postal directory:</p> <p><code>USHouseHoldDisplayBaseData=N</code></p> <p>Rooftop Census Geocoder DISPLAY_BASEDATA Values are listed in Volume II.</p>
USDPVParmFile	<p>Path that specifies the directory location and name of the US Delivery Point Validation (DPV) parameter file:</p> <p><code>USDPVParmFile=/myserver/tril7vnn/UCA_proj/parms/pfdpv.par</code></p>
USDPVEchoFile	<p>Path that specifies the directory location and name of the US DPV echo file, which is used for debugging:</p> <p><code>USDPVEchoFile=/myserver/tril7vnn/UCA_proj/data/pfdpvecho.txt</code></p>

Location of Libraries

There is a single library for the UCA that is named TrilEZCallableLibrary.

For Windows platforms, the .dll file of this library is located in the \bin folder of your Trillium Software System server installation.

Example: \trillium\tril7vnn\bin\TrilEZCallableLibrary.dll

For UNIX platforms, the appropriate library is located in the **/bin** folder of your Trillium Software System server installation.

Example: /trillium/tril7vnn/bin/TrilEZCallableLibrary.so

① *nn = Trillium Software System version number*

Class Files

Class files for calling from Java are located in the following Trillium Software server locations:

- \trillium\tril7vnn\bin\pkgs\natives
- \trillium\tril7vnn\bin\pkgs\rmi

① *nn = Trillium Software System version number*

Setting up the Sample UCA program

A test program is delivered in the folder that contains the sample projects and programs. This test is a Java program that can be used to test the installation and basic configuration of the UCA. It can also become a resource for understanding the methods that need to be called from a custom application.

The following instructions describe the modifications to the program and configuration files a programmer needs to make in order to execute the supplied sample program.

API Project Files

The UCA test program relies on a project delivered with the sample program. This project contains preconfigured parameter files and other resources necessary for the test to run. To use this project, create a project folder named **uca_proj** under your top level server software location. Copy the sample API projects from the samples location to this new project folder on your server.

Compiling and Executing the TestEZCallable.java Program

Compiling

1. Open the TestEZCallable.java file in your \uca_proj\bin folder and look for the following comments:
 - Comment out - `import pkgs.native.*;`
 - Uncomment - `//import pkgs.RMI.*;`
2. Make sure the **classpath** variable is in your \tril7vnn\bin folder, so it can find the correct class files.

Executing

If you are use RMI, then you can specify the host and port number from the command line of the calling application `-Dtrillium.host` and `-Dtrillium.port`.

If you do not specify these values, it will look for the `rmiserver.properties` file inside your home directory, which looks similar to the following:

```
port=1099
host=172.21.109.9 (IP address or hostname of the server machine)
```

In addition to RMI settings, the `TestEZCallable.java` is performing `System.getProperty` methods, so you will need to specify the following `-D` properties on the command line:

```
-Dparmfile = ..\parms\parmfile.ini
-Dfieldsfile = ..\parms\fieldnames.txt
-Ddatafile = ..\data\testEZCallableData
```

Modify the **`runtestezcallable.bat`** or **`runtestezcallable.sh`** elements to provide necessary `-D` properties.

UCA Methods Calling Order

The UCA is called by using the following subroutines; each call uses only the parameters it requires.

Table 2.2 Universal Cleansing Adapter Methods

Method	Description
open	Initializes all required functions listed in <i>parmfile.ini</i> .
process	Moves the input data string through the functions.
getField	<p>Retrieves data from a specified field by supplying the field name. Enables access to the output of different modules by field name (i.e. pr_first, pr_middle1, pr_gender).</p> <p>It also includes derived fields that are populated without concern for country of origin (such as dr_city, dr_state, or dr_postal_code).</p> <p>Call the getField method in a loop, once for each field you want retrieved from output.</p>
getFieldSeq	<p>Allows data to be retrieved from a field. Also for a field suffix to be used in the search.</p> <p>Ex: If pr_first_01 and pr_first_02 are filled, you can specify the field name <i>minus the suffix</i> (pr_first), and then enter the suffix in the seqNumber parameter (01 or 02).</p>
getFieldBytes	<p>Allows data to be retrieved from a field. Also allows for a field suffix to be used in the search. (If pr_first_01 and pr_first_02 are filled, you can specify the field name <i>minus the suffix</i> (pr_first), and then enter the suffix in the seqNumber parameter (01 or 02).</p> <p>The number of bytes is also returned, along with the value of this field.</p>

Table 2.2 Universal Cleansing Adapter Methods (Continued)

Method	Description
getFieldBytesSeq	<p>Allows data to be retrieved from a specified field by supplying the field name and sequence number. This method also allows for a field suffix to be used in the search.</p> <p>If pr_first_01 and pr_first_02 are both filled, you can specify the field name <i>minus the suffix</i> (pr_first), and then enter the suffix in the seqNumber parameter (01 or 02).</p> <p>The number of bytes is also returned, along with the value of this field.</p>
release	Releases the internal C memory created by a Process call.
getLastMessage	Returns descriptive error messages during Open or Process calls.
close	Terminates the program, closing all individual components opened in the Open call.

UCA Calling Methods Details

This section lists details for all of the UCA methods.

open Method

The **open** method opens the Router (if required) and all Parser(s) and Geocoder(s) listed in the UCA parameter file.

This table shows an **open** call method in each supported calling language:

Java	<pre>public int open(java.lang.String parmFile, int[] retCode)</pre>
C++	<pre>int __cdecl Open(char* parmFile, long* openHandle);</pre>
VB .NET	<pre>Declare Ansi Function Open Lib "trilEZCallableLibrary.dll" _ (ByVal parmFile As String, _ ByRef openHandle As Integer) As Integer</pre>
C#	<pre>[DllImport ("trilEZCallableLibrary.dll", EntryPoint = "Open", CharSet = CharSet.Ansi)] public static extern int Open(string parmFile, ref int openHandle);</pre>

This table describes the parameters used by the **open** method.

Table 2.3 UCA open Method Parameters

Parameter	Type	Use	Length	Description
parmFile	string (<i>Java</i> , <i>C#</i> , <i>VB .NET</i>) char* (<i>C++</i>)	In	variable	Name and location of <i>parmfile.ini</i> . Contains the list of Parsers and Geocoders used in the UCA implementation.
retCode	int (<i>Java</i>)	Out	Up to 4	Status code returned by the method. <i>See these codes in "UCA Return Codes" in Chapter 4 of Volume II.</i>
openHandle	long (<i>C++</i>) Integer (<i>VB .NET</i>) int (<i>C#</i>)	Out	-	Handle returned by the Open call that is used in subsequent calls.

Return Values

For C++, C# and VB. NET, the function status code (return code) is returned.

For Java, the following values can be returned:

- 1 Error; could not successfully execute.
- >= 0 Success (Open Handle).

process Method

The **process** method processes a data string through the Global Data Router, Parser, and Geocoder for a specific country. It returns a handle that is used in subsequent calls that are associated with this particular process call.

The following table describes a call to the **process** method in each supported calling language:

```
Java      public long process(int openHandle,
                       byte[] inArea,
                       int[] retCode)
```

In Java, use this process method when your data contains accented characters.

```
C++      int __cdecl Process(char* inArea[1000],
                       long* openHandle,
                       long* procHandle);
```

```
VB .NET  Declare Ansi Function Process Lib "trileZCallableLibrary.dll" _
         (ByVal inArea As String, _
         ByVal openHandle As Integer, _
         ByRef procHandle As Integer, _
         As Integer)
```

```
C#      [DllImport ("trileZCallableLibrary.dll",
                 EntryPoint = "Process",
                 CharSet = CharSet.Ansi)]
public static extern int Process(byte[] inArea,
                                int openHandle,
                                ref int procHandle);
```

This table describes the parameters used by the Process method.

Table 2.4 Process Method Parameters

Parameter	Type	Use	Length	Description
openHandle	int (Java, C#) long (C++) Integer (VB .NET)	In	-	Handle returned during the open call.
inArea	string (C# & VB .NET) char* (C++) byte [] (Java)	In	Up to 1000	Input data string processed through the UCA. Format is defined through the parser parameter file. All parameter files <i>must</i> have the input defined using the same shape.
retCode	int (Java)	Out	Up to 4	Status code returned by the method. <i>See these codes in the section "Universal Cleansing Adapter (UCA) Return Codes" in Chapter 4 of Volume II.</i>
procHandle	int (C#) long (C++) Integer (VB .NET)	Out	-	Handle created by the process call that is used in subsequent calls.

Return Values

For C++, C# and VB. NET, the function status code (return code) is returned.

For Java, the following values are returned:

-1 Error; could not successfully execute.

If >=0 Success (Process Handle).

getFieldBytes Method (Java only)

The **getFieldBytes** method retrieves data contained in an output field by supplying the field name. The number of bytes is returned, along with the data.

Use this method when your data contains accented characters.

This table describes a call to the **getFieldBytes** method in Java only:

```
Java      public java.lang.String getFieldBytes(long processHandle,  
                                           java.lang.String fieldName,  
                                           int[] retCode)
```

Table 2.5 getFieldBytes Method Parameters

Parameter	Type	Use	Length	Description
processHandle	long	In	–	Handle returned during the process call.
fieldName	string	In	variable	Contains a valid field name. See fldnames.txt for a list of field names.
retCode	int	Out	Up to 4	Status code returned by the method. See <i>these codes in the section "Universal Cleansing Adapter (UCA) Return Codes" in Chapter 4 of Volume II.</i>

Return Values

In Java, the contents of the specified field are what is returned.

getFieldBytesSeq Method (Java only)

The **getFieldBytesSeq** method retrieves the data contained in an output field by supplying the field name and sequence number. The number of bytes is also returned, along with the data.

Use this method when your data contains accented characters.

This table describes a call to the **getFieldBytesSeq** method in Java only:

```
Java      public java.lang.String getFieldBytesSeq(long processHandle,
                                             java.lang.String fieldName,
                                             int seqNumber,
                                             int[] retCode)
```

Table 2.6 getFieldBytesSeq Method Parameters

Parameter	Type	Use	Length	Description
processHandle	long	In	–	Handle returned during the process call.
fieldName	string	In	variable	Contains a valid field name. See fldnames.txt for a valid list of field names.
seqNumber	int	In	variable	Contains a field suffix to aid in the search. If pr_first_01 and pr_first_02 are both filled, you can specify the field name <i>minus the suffix (pr_first)</i> , and then enter the suffix here (01 or 02).
retCode	int	Out	Up to 4	Status code returned by the method. See <i>these codes in the section "Universal Cleansing Adapter (UCA) Return Codes" in Chapter 4 of Volume II.</i>

Return Values

In Java, the contents of the specified field are what is returned.

getField Method

The **getField** method retrieves the data contained in an output field by supplying the field name.

This table describes a call to the **getField** method in each supported calling language:

Java	<pre>public java.lang.String getField(long processHandle, java.lang.String fieldName, int[] retCode)</pre>
C++	<pre>int __cdecl GetField(const char* fieldName, unsigned int index, const char** fieldData, unsigned int fieldLength), long processHandle);</pre>
VB .NET	<pre>Declare Ansi Function GetField Lib "trileZCallableLibrary.dll" _ (ByVal fieldName As String, _ ByVal index As Integer, _ ByRef fieldData As String, _ ByRef fieldLength As Integer, _ ByVal processHandle As Integer) _ As Integer</pre>
C#	<pre>[DllImport ("trileZCallableLibrary.dll", EntryPoint = "GetField", CharSet = CharSet.Ansi)] public static extern int GetField(string fieldName, int index, ref string fieldData, ref int fieldLength, int processHandle);</pre>

Table 2.7 getField Method Parameters

Parameter	Type	Use	Length	Description
processHandle	int (C#) long (C++, Java) Integer (VB .NET)	In	-	Handle returned during the process call.
fieldName	string (Java, C#, & VB .NET) char* (C++)	In	variable	Contains a valid field name. See fldnames.txt for a list of field names.
retCode	int (Java)	Out	Up to 4	Status code returned by the method. See <i>these codes in the section "Universal Cleansing Adapter (UCA) Return Codes" in Chapter 4 of Volume II.</i>
index	int (C++ & C#) Integer (VB .NET)	In	-	Allows segments of data to be retrieved by index. Up to 10 segments can be retrieved individually. Default is 0.
fieldData	string (VB NET & C#) char** (C++)	Out	variable	Contents of the retrieved field.
fieldLength	int (C++ & C#) Integer (VB .NET)	Out	-	Length of the field returned to the output.

Return Values

For C++, C#, and VB .NET, returns the function status code (return code).

For Java, the contents of the specified field are what is returned.

getFieldSeq Method

The **getFieldSeq** method retrieves the data contained in an output field by supplying the field name, minus the numeric suffix.

The suffix can be added into the seqNumber parameter to aid in searching. This table describes a call to the **getFieldSeq** method for JAVA:

```
Java    public java.lang.String getField(long processHandle,  
                                           java.lang.String fieldName,  
                                           int seqNumber,  
                                           int[] retCode)
```

Table 2.8 getFieldSeq Method Parameters

Parameter	Type	Use	Length	Description
processHandle	Long	In	-	Handle returned during the getFieldSeq call.
fieldName	String	In	variable	Contains the name of a valid field. <i>See the fldnames.txt file for a list of valid field names.</i>
seqNumber	int	In	variable	Contains a field suffix to aid in the search. If pr_first_01 and pr_first_02 are both filled, you can specify the field name <i>minus the suffix (pr_first)</i> , and then enter the suffix here (01 or 02).
retCode	int	Out	Up to 4	Status code returned by the method. <i>See these codes in the section "Universal Cleansing Adapter (UCA) Return Codes" in Chapter 4 of Volume II.</i>

Return Values

For Java, the contents of the specified field are what is returned.

release Method

The **release** method releases the internal 'C' memory created by the **Process** call.

This table describes a call to the **release** method in each supported calling language:

```

Java      public int release(long processHandle)
C++      int __cdecl Release(long processHandle);
VB .NET  Declare Ansi Function Release Lib
            "trilEZCallableLibrary.dll" _
            (ByVal processHandle As Integer)
            As Integer
C#      [DllImport ("trilEZCallableLibrary.dll",
                    EntryPoint = "Release",
                    CharSet = CharSet.Ansi)]
            public static extern int Release(int processHandle);

```

Table 2.9 Release Method Parameters

Parameter	Type	Use	Length	Description
processHandle	long (<i>Java & C++</i>) int (<i>C#</i>) Integer (<i>VB .NET</i>)	In	-	Handle returned by the process call that is used in subsequent calls.

Return Values

Returns a status value of 0.L

getLastMessage

The **getLastMessage** method displays a descriptive error message when an error occurs during an **open** or **process** call.

This table describes a call to the **release** method in each supported calling language:

```
Java      public java.lang.String getLastMessage(int openHandle)
C++      char* __cdecl GetLastMessage(long openHandle);
VB .NET  Declare Ansi Function GetLastMessage Lib "trileZCallableLibrary.dll" _
           (ByVal openHandle As Integer)
           As String
C#      [DllImport ("trileZCallableLibrary.dll",
                   EntryPoint = "GetLastMessage",
                   CharSet = CharSet.Ansi)]
           public static extern string GetLastMessage(int openHandle);
```

This table describes the parameters used by the `getLastMessage` method.

Table 2.10 `getLastMessage` Method Parameters

Parameter	Type	Use	Length	Description
openHandle	long (C++) int (<i>Java and C#</i>) Integer (<i>VB .NET</i>)	In	-	Handle returned by the open call that is used in subsequent calls.

Return Values

Returns a data string that contains a descriptive error message.

close Method

The **close** method closes **TrileZCallable**, which in turn closes all individual components opened in the **Open** call.

This table describes a call to the **close** method in each supported calling language:

```

Java      public int close(int openHandle)

C++      int __cdecl Close(int openHandle);

VB .NET  Declare Ansi Function Close Lib "trileZCallableLibrary.dll" _
            (ByVal openHandle As Integer)
            As Integer

C#      [DllImport ("trileZCallableLibrary.dll",
                    EntryPoint = "Close",
                    CharSet = CharSet.Ansi)]
            public static extern int Close(int openHandle);
  
```

This table describes the parameters used by the **close** method.

Table 2.11 close Method Parameters

Parameter	Type	Use	Length	Description
openHandle	long (C++) int (Java and C#) Integer (VB .NET)	In	-	Handle returned by the open call that is used in subsequent calls.

Return Values

For C++, C#, and VB .NET, the function status code (return code) is returned.

For Java, a status value of 0 is returned.

List of UCA-Derived Fields

The following table shows “derived” fields and their sizes for the UCA function.

A **derived field** can be the normalized name of a geocoder field, where some geocoders may have different spellings for the same field. It can also be a concatenation of multiple fields, such as address or city name.

The “origin” column shows the name of the PREPOS or geocoder field(s) from where the data is sourced. All other field names available to the calling application are from the Customer Data Parser or postal geocoder. Those field names can be found in the Programmer’s Guide Volume 2 for each module.

Table 2.12 UCA List of Derived Fields

“dr” field name	Length	Origin
Customer Data Parser PREPOS		
dr_prefix	15	prepos.name_seg.pr_prefix_display
dr_first_name	15	prepos.name_seg.pr_first_display
dr_middle_name	15	prepos.name_seg.pr_middle1_display
dr_last_name	30	prepos.name_seg.pr_last_display
dr_name	100	prepos.label_seg.pr_line
dr_city_name	30	prepos.geog_seg.pr_city_name
dr_st_prov_cty_name	30	prepos.geog_seg.pr_st_prov_cty_name
dr_house_number	15	prepos.street_seg.pr_hse_nbr
dr_street_name	80	prepos street pre direction, name, suffix and post direction
dr_country	2	router output
dr_failleveltext	30	generated from out_fail_level

Table 2.12 UCA List of Derived Fields

"dr" field name	Length	Origin
US GEOCODER		
dr_postal_code	9	us.full_postal_code
dr_city_name	30	us.postal_city_name
dr_st_prov_cty_name	2	prepos.geocode_seg.pr_gin_stprvcty_name
dr_address	50	prepos.geocode_seg.pr_gout_deliver_addr
dr_house_number	10	prepos.geocode_seg.pr_gout_house_number
dr_rec_type	1	us.record_type
CA GEOCODER		
dr_postal_code	6	ca.postal_code
dr_city_name	30	ca.area_name
dr_st_prov_cty_name	2	ca.prov_cnty
dr_address	50	ca.deliver_addr
dr_house_number	10	ca.house_number
dr_rec_type	1	ca.rec_type
UK GEOCODER		
dr_postal_code	7	uk.out_postal_out
dr_city_name	30	uk.ptwn_name
dr_st_prov_cty_name	30	uk.cnty_name
dr_house_number	10	uk.house_number
dr_address	239	specialized logic to assemble an address

Table 2.12 UCA List of Derived Fields

"dr" field name	Length	Origin
HK GEOCODER		
dr_city_name	30	hk.postal_city
dr_house_number	8	hk.house_number
dr_address	102	specialized logic to assemble an address
dr_rec_type	1	hk.rec_type
AU GEOCODER		
dr_postal_code	4	au.postal_code
dr_city_name	30	au.postal_city
dr_st_prov_cty_name	3	au.state
dr_house_number	10	au.house_number
dr_address	50	au.deliver_addr
dr_rec_type	1	au.rec_type
DE GEOCODER		
dr_postal_code	5	de.postal_code
dr_city_name	30	de.postal_city
dr_house_number	8	de.house_number
dr_address	50	de.deliver_addr
dr_rec_type	1	de.rec_type
IT GEOCODER		
dr_postal_code	5	it.postal_code
dr_city_name	30	it.postal_city
dr_house_number	10	it.house_number
dr_address	50	it.deliver_addr

Table 2.12 UCA List of Derived Fields

"dr" field name	Length	Origin
dr_rec_type	1	it.rec_type
TG GEOCODER		
dr_postal_code	15	togo.postal_code
dr_city_name	50	togo.postal_city
dr_st_prov_cty_name	30	togo.state
dr_house_number	20	togo.house_number
dr_address	145	togo.deliver_addr
dr_rec_type	1	togo.rec_type

Trillium Software System API Calls on Windows

The Trillium Software System Applications Programming Interface (API) supports calls from the following languages on Windows:

- C/C++
- C#
- Visual Basic
- VB .NET
- Java

① *Paths set within a parameter file cannot exceed 80 characters.*

API Calls from C/C++ on Windows

The methods for calling the Trillium Software System modules from C or C++ on Windows are the same methods used on UNIX.

- ① *For complete information on each module, including library and directory information on the C or C++ platform, see Chapter 4, in the section "API Calls from C on UNIX" on page 4- 2.*

API Calls from C# on Windows

The Trillium Software System API supports calls from C#. To dynamically link to the Trillium Software System APIs, C# uses the DLLs located in the `.\cdll` directory that are installed with the Trillium Software System.

In order to use these DLL functions, the managed code must create prototypes for the unmanaged functions. In C#, apply the **DllImport** attribute and the `static` and `extern` modifiers to the function's prototype.

Call types for the 'Action' Parameter

There are three different call types as detailed by the **Action** parameter. Each call uses the same parameters with a different **Action** value. The **Action** parameter tells the program what function to perform for a specific call. The settings for the **Action** parameter are described in the following table.

- O Open** Initializes the module, loads required tables into memory, and reads the parameter (parm) file. For the Geocoder, it Creates the control area, opens Level 1 index into memory and sets up control areas from parameters. Also opens Level 2 and BaseData files and saves the file handles in the control area.
- P Process** Processes data in the Input buffer and writes data to the Output buffers. For the Geocoder, Uses control areas and its variables; all other variables are automatic (unique copies to each function). Reads Level 2 and Basedata files as needed and applies matching logic. Also reads the memory copy of Level 1.
- R Minimal Open** For the *Postal Geocoder(s) only*, functions the same as Process, except uses an automatic version of the Level 2 and Basedata file handles. (Performs its own file open/close on these two resource files.) *Required for multi-threaded use of Geocoder.*
- C Close** Closes the modules, cleans up allocated resources, and writes any statistical information.

The following DLLs are required to call the Trillium Software System modules:

C# uses this DLL:	To call this module:
cflib.dll	Converter
galib.dll	Business Data Parser
cflib.dll	Create Window Key
mllib.dll	Matcher (Reference Match, Window Match)

Calling the Converter API from C#

The following statement is required in order to use C# to call the Converter:

```
[DllImport ("cflib.dll",  
    EntryPoint = "cf_coni",  
    CharSet = CharSet.Ansi)]  
public static extern int cf_coni(string Action,  
    out IntPtr ControlArea,  
    string ParmFile,  
    string ParmEcho,  
    string InArea,  
    StringBuilder OutArea,  
    StringBuilder RetCode);
```

This table describes the parameters used by C# calls to the **cf_coni** function.

Parameter	Type	Use	Length	Description
Action	string	In	1	Defines the call type as either O=Open , P=Process , or C=Close .
ControlArea	IntPtr	In/ Out	16	Control work area created on the Open call, then used during Process and Close calls. If Action is either P or C=In ; if Action is O=Out .
ParmFile	string	In	variable	Name of parameter file.
ParmEcho	string	Out	variable	Name of parameter echo file used for debugging. If not used, set to Null. <i>Optional</i> .
InArea	string	In	Variable	Buffer that contains input data to process. Note that length should equal the record length specified in the input DDL.
OutArea	String Builder	Out	Variable	Buffer containing output data. Note that length should equal record length specified in output DDL.

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Return code value indicating status of the program call.

❗ *Error codes listed in Chapter 4 of the Programmer's Guide, Volume II.*

Calling the Business Data Parser API from C#

The following statement is required in order to use C# to call the Business Data Parser **ga_prsi** function:

```
[DllImport("galib.dll",
    EntryPoint = "ga_prsi",
    CharSet = CharSet.Ansi)]
public static extern void ga_prsi(string Action,
    StringBuilder ParmArea,
    string InArea,
    StringBuilder OutArea,
    string Null,
    string ParmFile,
    out IntPtr ControlArea);
```

This table describes the parameters used by C# calls to the **ga_prsi** function.

Parameter	Type	Use	Length	Description
Action	string	In	1	Defines the call type as either O=Open , P=Process , or C=Close .
ParmArea	String Builder	Out	10,000	Contains Business Data Parser parameter data.
InArea	string	In	1,000	Input buffer that contains the input data to be processed when the Process call is used.
OutArea	String Builder	Out	13,000	Buffer that contains output data. <i>See these codes in Chapter 4 of Volume II.</i>
NULL	string	-	4	A 4-byte pointer filled with null values (x'00').

3-6 Calling the Create Window Key API from C#

Parameter	Type	Use	Length	Description
ParmFile	string	In	variable	Name of parameter file.
ControlArea	IntPtr	In/ Out		Control work area created on the Open call, then used during the Process and Close calls. If Action is either P or C=In ; or if Action is O=Out .

Calling the Create Window Key API from C#

A *window key* is composed from elements in the input record. It is used to select records to include in the Match window for comparison. The program uses a rules-based parameter file to create up to 30 window keys per record. The parameter file will define the DDL and the rules file name that details the window key construction methods. This statement is required in order to use C# to call the **cf_crwkey** function:

```
[DllImport("cflib.dll",
    EntryPoint = "cf_crwkey",
    CharSet = CharSet.Ansi)]
public static extern void cf_crwkey(string Action,
    string ParmFile,
    string InputRecord,
    StringBuilder WindowKey,
    out IntPtr ControlArea);
```

This table describes the parameters used by C# calls to **cf_crwkey** function.

Table 3.1 Create Window Key (cf_crwkey) Parameters (C#)

Parameter	Type	Use	Length	Description
Action	string	In	1	Defines call type as O=Open , P=Process , or C=Close .
ParmFile	string	In	n*	Name of parameter file.

Table 3.1 Create Window Key (cf_crwkey) Parameters (C#) (Continued)

Parameter	Type	Use	Length	Description
InputRecord	string	In	variable	Record that contains the data used to derive the window key.
WindowKey	String Builder	Out	1500	Buffer holding window key results; can contain up to thirty 50-byte window keys. Keys are left-justified every 50 bytes.
ControlArea	IntPtr	In/ Out	16	Control work area built by the program on the Open call, then used during the Process and Close calls. If Action is either P or C=In ; if Action is O=Out .

n*= Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

Calling the Matcher API from C#

The Matcher differs from the other Trillium Software System modules because many calls to individual functions occur within the Matcher module itself. The match type defines which set of functions is required. There are two matching modes, *Reference Matching* and *Window Matching*; these matching modes are described in the following sections. For detailed information about Matcher functions, see the *Batch User's Guide*.

Reference Matching

This section discusses the steps required to use the Matcher in the Reference Match mode. This mode of operation matches one record, called the *candidate record*, to one or more records, called the *reference record*, that have been retrieved from a source such as a database. The match rules are contained in external parameter files that are loaded to memory upon initialization. The driver program *must* link to the library of Matcher routines.

Reference Match Requirements

The following requirements *must* be met for a Reference match:

- Candidate and reference record(s) *must* both have the same shape.
- Reference record(s) *must* be processed using the same or similar steps as the candidate record to ensure that the all records use standardized fields.
- Matcher Field List parameter files *must* be configured to use field locations (fldlocs) *not* DLL field names, as is commonly used in batch.
- If an optional candidate matching information key is used, reference records *must* contain a unique key for both household and individual levels.

Reference Match Step Names

This table defines the minimum steps required to perform a reference match and retrieve the results.

Table 3.2 Reference Match Step Names (C#)

Function Name	Description
m_initmr	Used to set up the Match windows and initialize the Matcher.
m_params	Call this function four times for a retail match; twice for a commercial match. Each call <i>must</i> process one of the required fields or patterns parameter files required for the Matcher. The field files <i>must</i> use the fldlocs keyword instead of the fldnames keyword because DLLs are not used in the calling process.
m_addrec	Used to load the Match window with reference records (the number of records with window key of the same value as the candidate record).
m_cmatch	Used to match the transaction record to the database records that have been loaded to the Match window during Add Record .
m_nummat	Call Number Reference Matches to obtain the number of records that matched to the candidate record.

Table 3.2 Reference Match Step Names (C#)

Function Name	Description
m_getmat	<p>Call Obtain Reference Match to return a value (in Number of Reference Matches) that is equal to the total number of reference match records that are available to be obtained. (One record is called at a time.)</p> <ul style="list-style-type: none"> • Each individual record written to a buffer. Records must be moved from the buffer before calling for another one. Otherwise, the original record is overwritten. • Buffer is the input record length plus 3 bytes holding the match pattern ID number (e.g. P150 would have a pattern ID of 150). • By default, the returned buffer is sorted in match pattern ID ascending order. If this order is not desired, the pattern IDs can all be set to the same value. • Calling parameters specify if you are requesting a household/commercial match or an individual match.
m_clearw	Used to empty the Match window, then restart with Add Record to match the next transaction.
m_endmat	After all matching is complete, this closes the Matcher and writes statistics to a file.

① *Required steps that are external to the Trillium Software System project, but required for the match process, are not included in this table. For example, a step which extracts records from a database to load to the Match window is an operation that is performed by either the calling program or another external subroutine.*

Window Matching

This section discusses the steps required to use the Matcher in the Window Match mode. This mode of operation matches a group of records to one another. The Match rules are contained in external parameter files that are loaded to memory upon initialization. The driver program *must* link to the library of Matcher routines.

Window Match Step Names and Module Names

The list provided in the table below is provided as an example of the minimum steps required to perform a Window Match and retrieve the results. Each step name is followed by the entry point name in parenthesis in the program. See the appropriate section in the documentation that describes each module.

- ① *Required steps that are external to the project, but required for the match process, are not included in the table. For example, a step that extracts records from a database to load to the Match window is an operation that is performed by either the calling program or another external subroutine.*

This table defines the minimum steps required to perform a window match and retrieve the results.

Table 3.3 Window Match Function Names (C#)

Name	Description
m_initmr	Call Initialize Matcher to set up the Match windows and initialize the Matcher.
m_params	Call Process Parameters four times for a retail match; twice for a commercial match. Each call <i>must</i> process one of the required fields or patterns parameter files required for the Matcher. The field files <i>must</i> use the fldlocs keyword instead of the fldnames keyword because DDLs are not used in the calling process.
m_addrec	Call Add Record to add to a Match window. For the number of records with the same candidate code.
m_wmatch	Call Match Window to match the records that have been loaded to the Match window during Add Record .
m_common	Call Create Common to append the common data and survivor flag information on the matched records. <i>Window match only. Optional.</i>
m_retrvr	Call Retrieve Records for the number of window records to retrieve the matched record with appended match information. Return buffer <i>must</i> equal the input record length + 184 bytes.

Table 3.3 Window Match Function Names (C#) (Continued)

Name	Description
m_clearw	Call Clear Window to empty the Match window and restart with Add Record to match the next candidate code window.
m_endmat	Call End Matcher after all matching is complete to close the Matcher and write statistics to a file.

Initialize Matcher (m_initmr)

The **Initialize Matcher (m_initmr)** function initializes the Matcher by creating the Match window areas for either a Reference or Window match. The following statement is required to use C# to call the **m_initmr** function:

```
[DllImport("mllib.dll",
    EntryPoint = "m_initmr",
    CharSet = CharSet.Ansi)]
public static extern void m_initmr(StringBuilder RetCode,
    string StatFile,
    string LinkFile,
    ref int RecLen,
    out IntPtr
    ControlArea);
```

3-12 Initialize Matcher (*m_initmr*)

This table describes parameters used by C# calls to the **m_initmr** function.

Table 3.4 Initialize Matcher (*m_initmr*) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Error code returned by the subroutine. <i>See these error codes in Chapter 4 of Volume II.</i>
StatFile	string	In	n*	Name of file to write match statistics.
LinkFile	string	In	n*	Name of file to contain keys of matched records when performing multiple window match scenarios.
RecLen	int	In		Length of the input records to match. A negative record length activates debug ; writes data to the statistics file.
ControlArea	IntPtr	Out	16	Address of parameter area that resides in the API program and maintained by the Matcher. Created during Initialize Matcher, passed back to program; <i>must</i> be passed to each subsequent call.

n*= Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

Process Parameters (m_params)

The **Process Parameters (m_params)** function processes a Matcher parameter file (either a field list or a pattern list). The following statement is required in order to use C# to call the **m_params** function:

```
[DllImport("mllib.dll",
    EntryPoint = "m_params",
    CharSet = CharSet.Ansi)]
public static extern void m_params(StringBuilder RetCode,
    string ParmFileName,
    string ParmType,
    string Handle,
    string Rname,
    out IntPtr ControlArea);
```

Table 3.5 Process Parameters (m_params) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Error code returned by the subroutine. See <i>"Error Codes for Process Parameters"</i> in <i>Chapter 4 of Volume II</i> .
ParmFileName	string	In	n*	Name of the matcher parameter file to process.
ParmType	string	In	1	Numerical indicator of the matcher parameter file to process. See next table.
Handle	string	In	--	Reserved for future use. Pass in Nulls.
Rname	string	In	--	Reserved for future use. Pass in Nulls.
ControlArea	IntPtr	In	16	Address of parameter area in API program. Maintained by the Matcher. Created during Initialize Matcher, passed back to program; <i>must</i> be passed to each subsequent call.

n*= Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

3-14 Add Record (m_addrec)

The **Add Record (m_addrec)** function loads the Match window with records.

Add Record (m_addrec)

This statement is required in order to use C# to call the **m_addrec** function:

```
[DllImport("mllib.dll",
    EntryPoint = "m_addrec",
    CharSet = CharSet.Ansi)]
public static extern void m_addrec
    (StringBuilder RetCode,
    string WindowType,
    string TransactionRecord,
    out IntPtr ControlArea);
```

This table describes parameters used by C# calls to the **m_addrec** function.

Table 3.6 Add Record (m_addrec) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	See the section "Error Codes for Add Record" in Chapter 4 of Volume II.
WindowType	string	In	1	Indicates the window to add the record; either R=Retail or C=Commercial .
TransactionRecord	string	In	variable	Record to add. Length <i>must</i> equal record length defined in the initializeMatcher call.
ControlArea	IntPtr	In	16	Address of parameter area that resides in the calling program and is maintained by the Matcher. Area is created during Initialize Matcher, passed back to the program, and <i>must</i> be passed to each subsequent call.

Clear Window (m_clearw)

The **Clear Window (m_clearw)** function erases the specified window record list and can optionally produce window statistics. The following statement is required in order to use C# to call the **m_clearw** function:

```
[DllImport("mllib.dll",
    EntryPoint = "m_clearw",
    CharSet = CharSet.Ansi)]
public static extern void m_clearw(StringBuilder RetCode,
    string WindowType,
    out IntPtr
    ControlArea);
```

This table describes parameters used by C# calls to the **m_clearw** function.

Table 3.7 Clear Window (m_clearw) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Error code returned. See "Error Codes for Clear Window" in Chapter 4 of Volume II.
WindowType	string	In	1	Indicates the window to add record; either R=Retail or C=Commercial .
ControlArea	IntPtr	In	16	Address of parameter area that resides in the API program and maintained by the Matcher. Area is created during Initialize Matcher, passed back to the API program, and <i>must</i> be passed to each subsequent call.

Match Candidate (m_cmatch)

The **Match Candidate (m_cmatch)** function initiates the matching of a specified candidate record against records in the specified window record list and saves information about the best matches (*Reference match*). The following statement is required for C# to call the **m_cmatch** function:

```
[DllImport("mllib.dll",
    EntryPoint = "m_cmatch",
    CharSet = CharSet.Ansi)]
public static extern void m_cmatch(StringBuilder RetCode,
    string WindowType,
    string CandidateRecord,
    out IntPtr
    ControlArea);
```

This table describes parameters used by C# calls to the **m_cmatch** function.

Table 3.8 Match Candidate (m_cmatch) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	See the section "Error Codes for Match Candidate" in Chapter 4 of Volume II.
WindowType	string	In	1	Indicates the window to match candidate record; either R=Retail or C=Commercial .
CandidateRecord	string	In	variable	Buffer that contains candidate record to match to the records in the window. Length used here must be the same as what was defined during the InitializeMatcher call.
ControlArea	IntPtr	In	16	Address of parameter area that resides in the calling program and maintained by the Matcher. Area is created during Initialize Matcher, passed back to the calling program, and <i>must</i> be passed to each subsequent call.

Number of Reference Matches (m_nummat)

The **Number of Reference Matches (m_nummat)** function finds the number of matches that occurred on a reference match. The following statement is required in order to use C# to call the **m_nummat** function:

```
[DllImport("mlib.dll",
    EntryPoint = "m_nummat",
    CharSet = CharSet.Ansi)]
public static extern void m_nummat(StringBuilder RetCode,
    string MatchLevel,
    out int NumHhldMats,
    out IntPtr
    ControlArea);
```

This table describes parameters used by C# calls to the **m_nummat** function.

Table 3.9 Number of Reference Matches (m_nummat) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Error code returned by the subroutine. See these codes listed in Chapter 4 of Volume II.
MatchLevel	string	In	1	Indicator for match level; either 0=Household or 1=individual . A <i>Commercial</i> match is the same as a <i>Household</i> match, and therefore uses a 0 value.
NumHhldMats	int	Out	-	Resulting number of matches in the window.
ControlArea	IntPtr	In	16	Address of the parameter area residing in the calling program and maintained by the Matcher. Area is created during Initialize Matcher, passed back to the calling program, and <i>must</i> be passed to each subsequent call.

Number of Reference Suspects (m_numsus)

The **Number of Reference Suspects (m_numsus)** function finds the number of suspect matches that occurred on a reference match. The following statement is required in order to use C# to call the **m_numsus** function:

```
[DllImport("mllib.dll",
    EntryPoint = "m_numsus",
    CharSet = CharSet.Ansi)]
public static extern void m_numsus(StringBuilder RetCode,
    string MatchLevel,
    out int NumSusps,
    out IntPtr
    ControlArea);
```

This table describes parameters used by C# calls to the **m_numsus** function.

Table 3.10 Number of Reference Suspects (m_numsus) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Error code returned by the subroutine. See these codes listed in Chapter 4 of Volume II.
MatchLevel	string	In	1	Indicator for match level; 0=Household or 1=Individual . A <i>Commercial</i> match is the same as a <i>Household</i> match, and uses a 0 value.
NumSusps	int	Out	--	Resulting number of suspect matches in the window.

**Table 3.10 Number of Reference Suspects (m_numsus) Parameters (C#)
(Continued)**

Parameter	Type	Use	Length	Description
ControlArea	IntPtr	In	16	Address of parameter area residing in the calling program and maintained by the Matcher. Area is created during Initialize Matcher, passed back to the API program, and <i>must</i> be passed to each subsequent call.

Obtain Reference Match (m_getmat)

The **Obtain Reference Match (m_getmat)** function obtains the specified record that matched successfully on a reference match from the specified window. This statement is required to use C# to call the **m_getmat** function:

```
[DllImport("mllib.dll",
    EntryPoint = "m_getmat",
    CharSet = CharSet.Ansi)]
public static extern void m_getmat(StringBuilder RetCode,
    string MatchLevel,
    ref int WhichMatch,
    StringBuilder Buffer,
    out IntPtr
    ControlArea);
```

This table describes the parameters used by the **m_getmat** function.

Table 3.11 Obtain Reference Match (m_getmat) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Error code returned by the subroutine. See <i>these codes listed in Chapter 4 of Volume II.</i>

Table 3.11 Obtain Reference Match (m_getmat) Parameters (C#)

Parameter	Type	Use	Length	Description
MatchLevel	string	In	1	Indicator for match level; either 0=Household or 1=Individual . A <i>Commercial</i> match is the same as a <i>Household</i> match, and therefore uses a 0 value.
WhichMatch	int	In	-	Numerical indicator of that matched record to retrieve.
Buffer	String Builder	Out	variable	Buffer to receive the record from the window. Length must be the record length plus three bytes (lrecl + 3) to hold the pattern ID.
ControlArea	IntPtr	In	16	Address of parameter area in the API program and maintained by the Matcher. Area is created during Initialize Matcher, passed back to the API program, and <i>must</i> be passed to each subsequent call.

Obtain Reference Suspect (m_getsus)

The **Obtain Reference Suspect (m_getsus)** function obtains the specified record that suspected successfully on a reference match from the specified window. This statement is required to use C# to call the **m_getsus** function.

```
[DllImport("mllib.dll",
    EntryPoint = "m_getsus",
    CharSet = CharSet.Ansi)]
public static extern void m_getsus(StringBuilder RetCode,
    string MatchLevel,
    ref int WhichSuspect,
    StringBuilder Buffer,
    out IntPtr
    ControlArea);
```

This table describes the parameters used by the **m_getsus** function.

Table 3.12 Obtain Reference Suspect (m_getsus) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Error code returned by the subroutine. See these codes listed in Chapter 4 of Volume II.
MatchLevel	string	In	1	Indicator for match level; either 0=Household or 1=Individual . A <i>Commercial</i> match is the same as a <i>Household</i> match, and therefore uses a 0 value.
WhichSuspect	int	In	variable	Numeric indicator of suspect matched record to retrieve.
Buffer	String Builder	Out	variable	Buffer to receive the record from the window. Buffer length must be the record length plus three bytes (record length + 3) to hold the matched pattern ID.
ControlArea	IntPtr	In	16	Address of parameter area that resides in the calling program and maintained by the Matcher. Area is created during Initialize Matcher, passed back to the calling program, and <i>must</i> be passed to each subsequent call.

End Matcher (m_endmat)

The **End Matcher (m_endmat)** function can optionally display the program statistics. This function erases both the retail and commercial window record lists, if they still exist, and terminates the Matcher. The following statement is

3-22 End Matcher (m_endmat)

required in order to use C# to call the (**m_endmat**) function:

```
[DllImport("mlib.dll",
    EntryPoint = "m_endmat",
    CharSet = CharSet.Ansi)]
public static extern void m_endmat(StringBuilder RetCode,
    out IntPtr
    ControlArea);
```

This table describes parameters used by C# calls to the **m_endmat** function.

Table 3.13 End Matcher (m_endmat) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Error code returned by the subroutine. <i>See these error codes listed in Chapter 4 of Volume II.</i>
ControlArea	IntPtr	In	16	Address of parameter area that resides in the calling program and maintained by the Matcher. Area is created during Initialize Matcher, passed back to the calling program, and <i>must</i> be passed to each subsequent call.

Create Common (m_common)

The **Create Common (m_common)** function (*Window matching only*) creates the common data segment for matched records in a record list. This function can optionally produce the linked records file. Common data is built of the “best” data from matches, based on the rules defined in the common fields parameter file.

The following statement is required in order to use C# to call the **m_common** function:

```
[DllImport("mlib.dll",
    EntryPoint = "m_common",
    CharSet = CharSet.Ansi)]
public static extern void m_common(StringBuilder RetCode,
    string WindowType,
    out IntPtr
    ControlArea);
```

This table describes parameters used by C# calls to the **m_common** function.

Table 3.14 Create Common (m_common) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Return code. <i>See these error codes listed in Chapter 4 of Volume II.</i>
WindowType	string	In	1	Indicates the window from which the common data segment is generated: R =Retail C =Commercial
ControlArea	IntPtr	In	16	Control work area used by the Matcher.

Match Window (*m_wmatch*)

The **Match Window (*m_wmatch*)** function initiates the matching of a specified window record list against itself. The ***m_wmatch*** function then appends the results of the matching process to each record in the record list (*Window match only*).

The following statement is required in order to use C# to call the ***m_wmatch*** function:

```
[DllImport("mlib.dll",
           EntryPoint = "m_wmatch",
           CharSet = CharSet.Ansi)]
public static extern void m_wmatch(StringBuilder RetCode,
                                   string WindowType,
                                   out IntPtr
                                   ControlArea);
```

This table describes the parameters used by C# calls to the ***m_wmatch*** function.

Table 3.15 Match Window (*m_wmatch*) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Return code. <i>See these error codes listed in Chapter 4 of Volume II.</i>
WindowType	string	In	1	Indicates the window to use in the match, and therefore, the type of matching: R =Retail C =Commercial
ControlArea	IntPtr	In	16	Control work area used by Matcher.

Sort Window (m_sortwn)

The **Sort Window (m_sortwn)** function sorts the matched window records into a specified order (for example, matched individuals in matched household.) The following statement is required in order to use C# to call the **m_sortwn** function:

```
[DllImport("mllib.dll",
           EntryPoint = "m_sortwn",
           CharSet = CharSet.Ansi)]
public static extern void m_sortwn(StringBuilder RetCode,
                                   string WindowType,
                                   string SortOrder,
                                   out IntPtr
                                   ControlArea);
```

This table describes parameters used by C# to call the **m_sortwn** function.

Table 3.16 Sort Window (m_sortwn) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Sort return codes. <i>See these error codes listed in Chapter 4 of Volume II.</i>
WindowType	string	In	1	Indicates the type of Matching window to use in the match; either R=Retail or C=Commercial .
SortOrder	string	In	1	Indicates which keys are used for the sort: -Matched individuals within matched households -Suspect individuals within suspect households -Matched individuals within suspect individuals within matched households
ControlArea	IntPtr	In	16	Control work area used by the Matcher.

Retrieve Record (m_retrvr)

The **Retrieve Record (m_retrvr)** function retrieves the specified relative record from a specified window record list.

The following statement is required in order to use C# to call the **m_retrvr** function:

```
[DllImport("mlib.dll",
    EntryPoint = "m_retrvr",
    CharSet = CharSet.Ansi)]
public static extern void m_retrvr(StringBuilder RetCode,
    string WindowType,
    ref int RecNum,
    StringBuilder WinRec,
    out IntPtr ControlArea);
```

This table describes parameters used by C# calls to the **m_retrvr** function.

Table 3.17 Retrieve Record (m_retrvr) Parameters (C#)

Parameter	Type	Use	Length	Description
RetCode	String Builder	Out	1	Return code. <i>See these error codes listed in Chapter 4 of Volume II.</i>
WindowType	string	In	1	Indicates which window to use: R =Retail C =Commercial
RecNum	int	In	variable	Specifies the record to retrieve.

Table 3.17 Retrieve Record (m_retrvr) Parameters (C#) (Continued)

Parameter	Type	Use	Length	Description
WinRec	String Builder	Out	variable	Contains the retrieved record. If window matching (Window Match) has been used to match records, the length is the input length + 184 bytes.
ControlArea	IntPtr	In	16	Control work area used by Matcher.

API Calls from VB .NET

Trillium Software APIs are written in C; therefore, they *must* be declared in a module before you can call them. The following sections detail how to call the Trillium Software System modules from Visual Basic (VB) .NET. The VB .NET calls use the libraries in the `.\stdll` directory that are installed with the Trillium Software System. This table lists the DLLs used by Visual Basic .NET.

Table 3.18 DLLs Used by VB .NET Calls

VB .NET uses this DLL:	To call:
cflibstd.dll	Investigator
galibstd.dll	Business Data Parser
cflibstd.dll	Create Window Key
mllibstd.dll	Matcher (Window Match, Reference Match)

The call types for the 'Action' Parameter

There are three different call types as detailed by the **Action** parameter. Each call uses the same parameters with a different **Action** value. The **Action** parameter tells the program what function to perform for a specific call. The settings for the **Action** parameter are described in the following table.

- | | | |
|----------|---------------------|---|
| O | Open | Initializes the module, loads required tables into memory, and reads the parameter (parm) file. For the Geocoder, it Creates the control area, opens Level 1 index into memory and sets up control areas from parameters. Also opens Level 2 and BaseData files and saves the file handles in the control area. |
| P | Process | Processes data in the Input buffer and writes data to the Output buffers. For the Geocoder, Uses control areas and its variables; all other variables are automatic (unique copies to each function). Reads Level 2 and Basedata files as needed and applies matching logic. Also reads the memory copy of Level 1. |
| R | Minimal Open | For the Postal Geocoder(s) only , functions the same as Process, except uses an automatic version of the Level 2 and Basedata file handles. (Performs its own file open/close on these two resource files.)
<i>Required for multi-threaded use of Geocoder.</i> |
| C | Close | Closes the modules, cleans up allocated resources, and writes any statistical information. |

Calling the Converter API from VB .NET

The following statement is required in order to use VB .NET to call the Converter:

```
Declare Function Convert Lib "cflibstd.dll" Alias "_cf_coni@28" _
    (ByVal Action As String, _
     ByVal ControlArea As String, _
     ByVal ParmFile As String, _
     ByVal ParmEcho As String, _
     ByVal Input As String, _
     ByVal Output As String, _
     ByVal RetCode As String) As Integer
```

This table describes parameters used by VB .NET calls to the Converter. For more information about Converter functions, see the *Batch User's Guide*.

Table 3.19 Converter Parameters (VB .NET)

Parameter	Type	Use	Length	Description
Action	String	1	In	Defines the call type; O=Open , P=Process , or C=Close .
ControlArea	String	16	In/ Out	Control work area created on the Open call; used during Process and Close calls. Note that the buffer <i>must</i> be initialized with spaces before Open . If Action is either P or C=In ; if Action is O=Out .
ParmFile	String	variable	In	Name of the Converter parameter file.
ParmEcho	String	variable	Out	Name of the parameter echo file; used for debugging. <i>Optional</i> .
Input	String	variable	In	Buffer that contains input data to process; length should equal record length specified in the input DDL.

Table 3.19 Converter Parameters (VB .NET) (Continued)

Parameter	Type	Use	Length	Description
Output	String	variable	Out	Buffer that contains output data from the Process call; length should equal record length specified in the output DDL. Note that the buffer <i>must</i> be initialized with spaces before Process .
RetCode	String	1	Out	Return code from program execution. Note that the buffer <i>must</i> be initialized with spaces before Open .

Calling the Business Data Parser API from VB .NET

The following statement is required in order to use VB .NET to call the Business Data Parser:

```

Declare Sub Parse Lib "galibstd.dll" Alias "_ga_prsi@28" _
    (ByVal Action As String, _
    ByVal ParmArea As String, _
    ByVal InArea As String, _
    ByVal PrePos As String, _
    ByVal NullPtr As String, _
    ByVal ParmFile As String, _
    ByVal ControlArea As String)
    
```

This table describes parameters used by VB .NET to call the **ga_prsi** function.

Table 3.20 Business Data Parser (ga_prsi) Parameters (VB .NET)

Parameter	Type	Use	Length	Description
Action	String	In	1	Defines the call type as either O=Open , P=Process , or C=Close .
ParmArea	String	Out	10,000	Defined by Business Data Parser parameter data. Note that the buffer <i>must</i> be initialized with spaces before Open .

Table 3.20 Business Data Parser (ga_prsi) Parameters (VB .NET)

Parameter	Type	Use	Length	Description
InArea	String	In	1,000	Input data used by the Business Data Parser.
PrePos	String	Out	13,000	Data returned by the Parser; passed as input to the Add Mat function. Note that the buffer <i>must</i> be initialized with spaces before Process .
NullPtr	String		4	Null values (x'00').
ParmFile	String	In	variable	Name of the parameter file.
ControlArea	String	In/ Out	16	Control work area created on the Open call; also used during Process and Close calls. Note that the buffer <i>must</i> be initialized with spaces before Open . If Action is either P or C=In ; if Action is O=Out .

Calling the Create Window Key API from VB .NET

The Create Window Key function can be used to create the window keys used in matching. A *window key* is constructed from elements in the input record, which is then used to select records for inclusion to the Match window for comparison. The Create Window Key function uses a rules-based parameter file to create up to 30 window keys per record. The following Declare statement is required to use VB .NET to call the Create Window Key function:

```
Declare Sub Winkey Lib "cflibstd.dll" Alias "_cf_crwkey@20" _
    (ByVal Action As String, _
     ByVal ParmFile As String, _
     ByVal InputRecord As String, _
     ByVal WindowKey As String, _
     ByVal ControlArea As String)
```

This table describes VB .NET parameters used to call Create Window Key.

Table 3.21 Create Window Key (cf_crwkey) Parameters (VB .NET)

Parameter	Type	Use	Length	Description
Action	String	In	1	Defines call type as either O=Open , P=Process , or C=Close .
ParmFile	String	In	variable	Name of the parameter file.
InputRecord	String	In	variable	Record containing data used to derive the window key.
WindowKey	String	Out	1500	Buffer that holds up to thirty 50-byte window keys. Note that the buffer <i>must</i> be initialized with spaces before Open . Keys are left-justified every 50 bytes.
ControlArea	String	In/ Out	16	Control work area. Note that the buffer <i>must</i> be initialized with spaces before Open . If Action is either P or C=In ; if Action is O=Out .

 *For more information about creating window keys, see the Batch User's Guide.*

Calling the Matcher API from VB .NET

The Matcher differs from other modules because many calls to individual functions occur within the Matcher module itself. The match type defines which set of functions is required. There are two matching modes, **Reference Matching** and **Window Matching**. For detailed information about Matcher functions, see the *Batch User's Guide*.

Reference Matching

This section discusses the steps required to use the Matcher in the Reference Match mode. This mode of operation matches one record, called the *candidate record*, to one or more records, called the *reference record*, that have been retrieved from a source such as a database. The match rules are contained in external parameter files that are loaded to memory upon initialization. The driver program *must* link to the library of Matcher routines.

Reference Match Requirements

The following requirements *must* be met for a reference match:

- Candidate record and reference record(s) *must* both have the same shape.
- Reference record(s) *must* be processed using the same or similar steps as the candidate record to ensure that the all records use standardized fields.
- Matcher Field List parameter files *must* be configured to use field locations (**fldlocs**) not DLL field names, as is commonly used in batch.
- If an optional candidate matching information key is used, reference records *must* contain unique keys for both household and individual levels.

Reference Match Step Names

This table lists step names that can be used as an example of the minimum steps required to perform a reference match and retrieve the results. The step name is given followed by the entry point name in parenthesis in the program. Each module section of the documentation details its function.

- ① *Required steps external to the project, but required for the match process, are not included here. For example, a step that extracts records from a database to load to the Match window is an operation that is performed by either the calling program or another external subroutine.*

Table 3.22 Reference Match Step Names (VB .NET)

Name	Description
m_initmr	Call Initialize Matcher to set up the Match windows and initialize the Matcher.
m_params	Call Process Parameters four times for a retail match; twice for a commercial match. Each call <i>must</i> process one of the required fields or patterns parameter files required for the Matcher. The field files <i>must</i> use the fldlocs keyword instead of the fldnames keyword (DDLs are not used in the calling process.)

Table 3.22 Reference Match Step Names (VB .NET) (Continued)

Name	Description
m_addrec	Call Add Record to load the Match window with reference records. For the number of records with window key of the same value as the candidate record.
m_cmatch	Call Match Candidate to match the transaction record to the database records that have been loaded to the Match window during Add Record .
m_nummat	Call Number Reference Matches to obtain the number of records that matched to the candidate record.
m_getmat	Call Obtain Reference Match to return a value (in Number of Reference Matches) that is equal to the total number of reference match records that are available to be obtained. (One record is called at a time.) Each individual record is written to a buffer. That record must be moved from the buffer before calling for another record. Otherwise, original record is overwritten. Buffer is the input record length plus 3 bytes holding the match pattern ID number (e.g. P150 would have a pattern ID of 150). By default, the returned buffer is sorted in match pattern ID ascending order. If this order is not desired, the pattern ID numbers can all be set to the same value. Calling parameters specify if you are requesting a household/commercial match or individual match.
m_clearw	Call Clear Window to empty the Match window, then restart with Add Record to match next transaction.
m_endmat	Call End Matcher after all matching is complete; closes the Matcher and writes statistics to a file.

① *You can retrieve suspect matches in a similar way if you are using suspect match patterns. Call the **Number of Reference Suspects (m_numsus)** and use the value returned to loop for calling **Obtain Reference Suspects (m_getsus)**.*

Window Matching

The Window Match mode of operation matches a group of records to one another. The Match rules are contained in external parameter files that are loaded to memory upon initialization. The driver program *must* link to the library of Matcher routines.

Window Match Step Names and Module Names

The list provided in the table below is provided as an example of the minimum steps required to perform a Window Match and retrieve the results. Each step name is followed by the entry point name in parenthesis in the program. See the appropriate section in the documentation that describes each module.

- ① *Required steps that are external to the project, but required for the match process, are not included in the table.*

Table 3.23 Window Match Function Names (VB .NET)

Name	Description
m_initmr	Call Initialize Matcher to set up the Match windows and initialize the Matcher.
m_params	Call Process Parameters four times for a retail match; twice for a commercial match. Each call <i>must</i> process one of the required fields or patterns parameter files for the Matcher. The field files <i>must</i> use the fldlocs keyword instead of the fldnames keyword because DDLs are not used in the calling process.
m_addrec	Call Add Record to add to a Match window. For the number of records with the same candidate code.
m_wmatch	Call Match Window to match the records that have been loaded to the Match window during Add Record .
m_common	Call Create Common to append the common data and survivor flag information on the matched records. <i>Window match only. Optional.</i>
m_retrvr	Call Retrieve Records for the number of window records to retrieve the matched record with appended match information. Buffer <i>must</i> equal the input record length + 184 bytes.
m_clearw	Call Clear Window to empty the Match window and restart with Add Record .
m_endmat	Call End Matcher after all matching is complete to close the Matcher and write statistics.

The following Matcher functions can be called from VB .NET applications.

Initialize Matcher (m_initmr)

The **Initialize Matcher (m_initmr)** function creates the Match window areas for either a Reference match or a Window match. Use the following Declare statement to use VB .NET to call the **m_initmr** function:

```
Declare Sub InitMatcher Lib "mlibstd.dll" Alias "_m_initmr@20" _
    (ByVal RetCode As String, _
     ByVal StatFile As String, _
     ByVal LinkFile As String, _
     ByRef RecLen As Integer, _
     ByVal ControlArea As String)
```

This table describes parameters used for VB .NET calls to **m_initmr** function.

Table 3.24 Initialize Matcher (m_initmr) Parameters (VB .NET)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. Note that the buffer <i>must</i> be initialized with spaces. See the section "Error Codes from Initialize Matcher" in Chapter 4 of Volume II.
StatFile	String	In	variable	Name of the file that contains matching statistics.
LinkFile	String	In	variable	Name of the file to contain window keys of matched records when performing multiple window match scenarios. This file is only required when performing window matching.
RecLen	Integer	In	7	Length of the records to match. A negative length activates Matcher tracing.
ControlArea	String	Out	16	Control work area used by the Matcher. Note that the buffer <i>must</i> be initialized with spaces.

Process Parameters (m_params)

The **Process Parameters (m_params)** function processes a Matcher parameter file (either a field list or a pattern list). Use the following Declare statement to use VB .NET to call the **m_params** function:

```
Declare Sub ProcessParms Lib "mlibstd.dll" Alias "_m_params@24" _
    (ByVal RetCode As String, _
     ByVal ParmFileName As String, _
     ByVal ParmType As String, _
     ByVal Handle As String, _
     ByVal Rname As String, _
     ByVal ControlArea As String)
```

This table describes parameters used by VB .NET calls to the **m_params** function.

**Table 3.25 Process Parameters
(m_params) Parameters (VB .NET)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See the section "Error Codes from Process Parameters" in Chapter 4 of Volume II.
ParmName	String	In	variable	Name of parameter file to process.
ParmType	String	In	1	Type of parameter file to process. See the table below.
Handle	String	-	-	Reserved for future use. Pass nulls.
Rname	String	-	-	Reserved for future use. Pass nulls.
ControlArea	String	In	16	Control work area used by Matcher.

The possible values for the **ParmType** parameter are defined below.

Table 3.26 ParmType Parameter Values (VB .NET)

Value	Description
0	Field list parameters for producing Common Data Segment: branch_number middlename business_name lastname first_name postalcode gender street_title house_number source_id
<p><i>① The following field descriptions, along with dictionary field names or field positions and length values, should appear in the specified parameter file to allow for the creation of the common data segment.</i></p>	
1	Household field comparison routine list.
2	Individual field comparison routine list.
3	Business field comparison routine list.
4	Household grade pattern list.
5	Individual grade pattern list.
6	Business grade pattern list.

Add Record (m_addrec)

The **Add Record (m_addrec)** function loads the Match window with records. Use the following Declare statement to use VB .NET to call the **m_addrec** function:

```
Declare Sub AddRecord Lib "mlibstd.dll" Alias "_m_addrec@16" _
    (ByVal RetCode As String, _
     ByVal WindowType As String, _
     ByVal TransactionRecord As String, _
     ByVal ControlArea As String)
```

The parameters for the **m_addrec** function are described below.

**Table 3.27 Add Record
(m_addrec) Parameters (VB .NET)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. Note that the buffer <i>must</i> be initialized with spaces. See the section "Error Codes from Add Record" in Chapter 4 of Volume II.
WindowType	String	In	1	Indicates which window to expand; either R=Retail or C=Commercial .
TransactionRecord	String	In	variable	The record to add.
ControlArea	String	In	16	Control work area used by Matcher.

ClearWindow (m_clearw)

The **Clear Window (m_clearw)** function erases the specified window record list and produces window statistics. Use the following Declare statement to use VB .NET to call the **m_clearw** function:

```
Declare Sub ClearWindow Lib "mlibstd.dll" Alias "_m_clearw@12" _
    (ByVal RetCode As String, _
     ByVal WindowType As String, _
     ByVal ControlArea As String)
```

The parameters for the **m_clearw** function are described in the table below.

**Table 3.28 ClearWindow
(m_clearw) Parameters (VB .NET)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. <i>See the section "Error Codes from Clear Window" in Chapter 4 of Volume II.</i>
WindowType	String	In	1	Indicates the window to erase; either R=Retail or C=Commercial .
ControlArea	String	In	16	Control work area used by the Matcher.

Match Candidate (m_cmatch)

The **Match Candidate (m_cmatch)** function initiates the matching of a specified candidate record against records in the specified window record list, and saves information about the best matches. (*Reference match only*)

Use the following Declare statement to use VB .NET to call the **m_cmatch** function:

```
Declare Sub MatchCand Lib "mlibstd.dll" Alias "_m_cmatch@16" _
    (ByVal RetCode As String, _
     ByVal WindowType As String, _
     ByVal CandidateRecord As String, _
     ByVal ControlArea As String)
```

This table describes parameters used by VB .NET for calls to **m_cmatch**.

**Table 3.29 Match Candidate (m_cmatch)
Parameters (VB .NET)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Match return code. <i>See the section "Error Codes from Match Candidate" in Chapter 4 of Volume II.</i>
WindowType	String	In	1	Indicates the window to use for the match, and therefore, the type; either R=Retail C=Commercial.
CandidateRecord	String	In	variable	The candidate record to match.
ControlArea	String	In	16	Control work area used by the Matcher.

Number of Reference Matches (m_nummat)

The **Number of Reference Matches (m_nummat)** function finds the number of matches that occurred on a Reference match.

Use the following Declare statement to use VB .NET to call the **m_nummat** function:

```
Declare Sub NumRefMats Lib "mlibstd.dll" Alias "_m_nummat@16" _
    (ByVal RetCode As String, _
     ByVal MatchLevel As String, _
     ByRef NumMats As Integer, _
     ByVal ControlArea As String)
```

This table describes parameters used by VB .NET calls to the **m_nummat** function.

Table 3.30 Number of Reference Matches (m_nummat) Parameters (VB .NET)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See the section "Error Codes from Number of Reference Matches" in Chapter 4 of Volume II.
MatchLevel	String	In	1	Match level; either 0=Household or business ; or 1=Individual .
NumMats	Integer	Out	7	Number of matches.
ControlArea	String	In	16	Control work area used by Matcher.

Number of Suspect Matches (m_numsus)

The **Number of Suspect Matches (m_numsus)** function finds the number of suspects matches that occurred on a Reference match. Use the following Declare statement to use VB .NET to call the **m_numsus** function:

```
Declare Sub NumRefSusps Lib "mlibstd.dll" Alias "_m_numsus@16" _
    (ByVal RetCode As String, _
     ByVal MatchLevel As String, _
     ByRef NumSusps As Integer, _
     ByVal ControlArea As String)
```

The parameters for the **m_numsus** function are described in the table below.

Table 3.31 Number of Suspect Matches (m_numsus) Parameters (VB .NET)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See "Error Codes from Number of Suspect Matches" in Chapter 4 of Volume II.
MatchLevel	String	In	1	Match level; either: 0=Household or business , or 1=Individual .
NumSusps	Integer	Out	7	Number of suspects.
ControlArea	String	In	16	Control work area used by Matcher.

Obtain Reference Matches (m_getmat)

The **Obtain Reference Matches (m_getmat)** function obtains the specified record that matched successfully on a reference match from the specified window. The following Declare statement is required in order to use VB .NET to call the **m_getmat** function:

```
Declare Sub ObtainRefMat Lib "mlibstd.dll" Alias "_m_getmat@20" _
    (ByVal RetCode As String, _
     ByVal MatchLevel As String, _
     ByRef WhichMatch As Integer, _
     ByVal Buffer As String, _
     ByVal ControlArea As String)
```

This table describes parameters used for VB .NET calls to **m_getmat**.

Table 3.32 Obtain Reference Matches (m_getmat) Parameters (VB .NET)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See "Error Codes from Obtain Reference Matches" in Chapter 4 of Volume II.
MatchLevel	String	In	1	Match level, either 0=Household or business , or 1=Individual .
WhichMatch	Integer	In	n ¹	Match to retrieve.
Buffer	String	Out	n ²	Buffer to contain the record retrieved from the match window during the obtainRefMatch call. Buffer <i>must</i> be initialized with spaces.
ControlArea	String	In	16	Control work area used by Matcher.

n¹ = Length of numeric value indicating which match to retrieve.

n² = Length of the record as defined during the initializeMatcher call + 3 bytes to contain the matched pattern ID.

Obtain Reference Suspect Matches (m_getsus)

The **Obtain Reference Suspect Matches (m_getsus)** function obtains the specified record that suspected successfully on a Reference match from the specified window. The following Declare statement is required in order to use VB .NET to call the **m_getsus** function:

```
Declare Sub ObtainRefSusp Lib "mlibstd.dll" Alias "_m_getsus@20" _
    (ByVal RetCode As String, _
     ByVal MatchLevel As String, _
     ByRef WhichSuspect As Integer, _
     ByVal Buffer As String, _
     ByVal ControlArea As String)
```

The parameters for the **m_getsus** function are described below.

Table 3.33 Obtain Reference Suspect Matches (m_getsus) Parameters (VB .NET)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See "Error Codes from Obtain Reference Suspect Matches" in Ch. 4 of Volume II.
MatchLevel	String	In	1	Match level. Set to either: 0=Household/business , or 1=Individual .
WhichSuspect	Integer	In	n ¹	Specifies which suspect to retrieve.
Buffer	String	Out	n ²	Buffer to contain the record retrieved from the suspect match window during obtainRefMatch. Buffer <i>must</i> be initialized with spaces.
ControlArea	String	In	16	Control work area used by the Matcher.

n¹ = Length of numeric value indicating which suspect match to retrieve.

n² = Length of the record as defined during the initializeMatcher call + 3 bytes to contain the suspect pattern ID.

End Matcher (m_endmat)

The **End Matcher (m_endmat)** function displays program statistics, erases both Retail and Commercial window record lists, if they still exist, and terminates the Matcher. The following Declare statement is required in order to use VB .NET to call the **m_endmat** function:

```
Declare Sub EndMatcher Lib "mlibstd.dll" Alias "_m_endmat@8" _
    (ByVal RetCode As String, _
     ByVal ControlArea As String)
```

This table describes parameters used by VB .NET calls to the **m_endmat** function.

**Table 3.34 End Matcher
(m_endmat) Parameters (VB .NET)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See the section "Error Codes from End Matcher" in Chapter 4 of Volume II.
ControlArea	String	In	16	Control work area used by the Matcher.

Create Common (m_common)

The **Create Common (m_common)** function (*Window matching only*) creates the common data segment for matched records in a record list. This function can optionally produce the linked records file. Common data is built of the “best” data from matches, based on the rules defined in the common fields parameter file.

The following Declare statement is required in order to use VB .NET to call the **m_common** function:

```
Declare Sub CreateCommon Lib "mlibstd.dll" Alias "_m_common@12" _
    (ByVal RetCode As String, _
     ByVal WindowType As String, _
     ByVal ControlArea As String)
```

This table describes the parameters used by VB .NET calls to the **m_common** function.

**Table 3.35 Create Common
(m_common) Parameters (VB .NET)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See "Error Codes from Create Common" in Chapter 4 of Volume II.
WindowType	String	In	1	Indicates the window from which the common data segment is generated; either R=Retail or C=Commercial .
ControlArea	String	In	16	Control work area used by the Matcher.

Match Window (m_wmatch)

The **Match Window (m_wmatch)** function initiates the matching of a specified window record list against itself. The **m_wmatch** function then appends the results of the matching process to each record in the record list (*Window match only*).

The following Declare statement is required in order to use VB .NET to call the **m_wmatch** function:

```
Declare Sub MatchWindow Lib "mlibstd.dll" Alias "_m_wmatch@12" _
    (ByVal RetCode As String, _
     ByVal WindowType As String, _
     ByVal ControlArea As String)
```

The parameters for the **Match Window** function are described below.

**Table 3.36 Match Window
(m_wmatch) Parameters (VB .NET)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See the section "Error Codes from Match Window" in Chapter 4 of Volume II.
WindowType	String	In	1	Indicates the window to use in the match, and therefore, the type of matching, either R=Retail or C=Commercial .
ControlArea	String	In	16	Control work area used by Matcher.

Retrieve Record (m_retrvr)

The **Retrieve Record (m_retrvr)** function retrieves the specified relative record from a specified window record list. The following Declare statement is required in order to use VB .NET to call the **m_retrvr** function:

```
Declare Sub RetrieveRecord Lib "mlibstd.dll" Alias "_m_retrvr@20" _
    (ByVal RetCode As String, _
     ByVal WindowType As String, _
     ByRef RecNum As Integer, _
     ByVal WinRec As String, _
     ByVal ControlArea As String)
```

This table describes the parameters used by VB .NET calls to the **Retrieve Record** function.

Table 3.37 Retrieve Record (m_retrvr) Parameters (VB .NET)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. <i>See the section "Error Codes from Retrieve Record" in Chapter 4 of Volume II.</i>
WindowType	String	In	1	Indicates which window to use; either: R=Retail or C=Commercial .
RecNum	Integer	In	n ¹	Specifies the record to retrieve.
WinRec	String	Out	n ²	Contains the retrieved record. Note that the buffer <i>must</i> be initialized with spaces.
ControlArea	String	In	16	Control work area used by Matcher.

n¹ = Length of numeric value indicating which match to retrieve.

n² = Length of the record as defined during the initializeMatcher call + 184 bytes to contain the window match key results.

Sort Window (m_sortwn)

The **Sort Window (m_sortwn)** function sorts the matched window records into a specified order (for example, matched individuals in matched household).

The following Declare statement is required in order to use VB .NET to call the **m_sortwn** function:

```
Declare Ansi Sub SortWindow Lib "mlibstd.dll" Alias "_m_sortwn@16" _
    (ByVal RetCode As String, _
    ByVal WindowType As String, _
    ByVal SortOrder As String, _
    ByVal ControlArea As String)
```

This table describes parameters used for VB .NET calls to **m_sortwn** function.

**Table 3.38 Sort Window
(m_sortwn) Parameters (VB .NET)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Sort return codes. <i>See the section "Error Codes from Sort Window" in Chapter 4 of Volume II.</i>
WindowType	String	In	1	Indicates the type of Matching window to use in the match; either R=Retail or C=Commercial .
SortOrder	String	In	1	Indicates keys used for the sort; either: -Matched individuals within matched households -Suspect individuals within suspect households -Matched individuals within suspect individuals within matched households
ControlArea	String	In	16	Control work area used by the Matcher.

API Calls from Visual Basic

The Trillium Software System APIs are written in C; therefore, they *must* be declared in a module before you can call them. The following sections detail how to call the Trillium Software System modules from Visual Basic (VB). The VB calls use the dynamic-link libraries (DLLs) in the `.\stdll` directory that are installed with the Trillium Software System.

This table lists the DLLs used by Visual Basic applications.

Table 3.39 DLLs Used by VB Calls

VB uses this DLL:	To call:
cplibstd.dll	Converter
galibstd.dll	Business Data Parser
cplibstd.dll	Create Window Key
mllibstd.dll	Matcher (Window Match, Reference Match)

The VB call types for the 'Action' parameter

Visual Basic calls to the Trillium Software System modules are defined by the variables used with the **Action** parameter. These variables include:

- O Open** Initializes the module, loads required tables into memory, and reads the parameter (parm) file. Creates the control area, opens Level 1 index into memory and sets up control areas from parameters. Opens Level 2 and BaseData files and saves file handles in the control area.
- P Process** Processes data in the Input buffer and writes data to the Output buffers. Uses control areas and its variables, all other variables are automatic. Reads Level 2 and Basedata files as needed and applies matching logic. Reads the memory copy of Level 1.
- R Minimal Open** For the **Postal Geocoder(s) only**, functions the same as Process, except uses an automatic version of the Level 2 and Basedata file handles. (Performs its own file open/close on these two resource files.) *Required for multi-threaded use of Geocoder.*
- C Close** Closes the modules, cleans up allocated resources, and writes any statistical information.

Calling the Converter API from VB

The following Declare statement is required in order to use Visual Basic (VB) to call the Converter:

```
Declare Function Convert Lib "cflibstd.dll" Alias "_cf_coni@28" _
    (ByVal Action As String, _
     ByVal ControlArea As String, _
     ByVal ParmFile As String, _
     ByVal ParmEcho As String, _
     ByVal Input As String, _
     ByVal Output As String, _
     ByVal RetCode As String)
    As Long
```

This table describes parameters used by Visual Basic calls to the Converter. For complete information about Converter functions, see the *Batch User's Guide*.

Table 3.40 Converter Parameters (VB)

Parameter	Type	Use	Length	Description
Action	String	1	In	Defines call type as O=Open , P=Process , or C=Close .
ControlArea	String	16	In/ Out	Control work area created on the Open call; also used during Process and Close calls. Note that the buffer <i>must</i> be initialized with spaces before Open . If Action is either P or C=In ; if Action is O=Out .
ParmFile	String	variable	In	Name of the parameter file.
ParmEcho	String	variable	Out	Parameter echo file used for debugging. <i>Optional</i> .
Input	String	variable	In	Buffer that contains input data to process; length should equal record length specified in the input DDL.

Table 3.40 Converter Parameters (VB) (Continued)

Parameter	Type	Use	Length	Description
Output	String	variable	Out	Buffer containing output data from the Process call; length should equal record length specified in the output DDL. The buffer <i>must</i> be initialized with spaces before Process .
RetCode	String	1	Out	Return code from program execution. Note that the buffer <i>must</i> be initialized with spaces before Open .

Calling the Business Data Parser API from VB

The following Declare statement is required in order to use Visual Basic (VB) to call the Business Data Parser:

```
Declare Sub Parse Lib "galibstd.dll" Alias "_ga_prsi@28" _
    (ByVal Action As String, _
    ByVal ParmArea As String, _
    ByVal InArea As String, _
    ByVal PrePos As String, _
    ByVal NullPtr As String, _
    ByVal ParmFile As String, _
    ByVal ControlArea As String)
```

This table describes parameters used for VB calls to the Business Data Parser.

Table 3.41 Business Data Parser Parameters (VB)

Parameter	Type	Use	Length	Description
Action	String	In	1	Defines the call type: O=Open , P=Process , or C=Close .
ParmArea	String	Out	10,000	Defined by Business Data Parser parameter data. Note that the buffer <i>must</i> be initialized with spaces before Open .
InArea	String	In	1,000	Input data used by the Business Data Parser.
PrePos	String	Out	13,000	Output data from the Business Data Parser. Buffer <i>must</i> be initialized with spaces before Process . See the <i>Business Data PREPOS</i> section in Chapter 4 of Volume II.
NullPtr	String	-	4	Null values (x'00').
ParmFile	String	In	variable	Name of the parameter file

Table 3.41 Business Data Parser Parameters (VB) (Continued)

Parameter	Type	Use	Length	Description
ControlArea	String	In/ Out	16	Control work area created on the Open call; also used during Process and Close calls. If Action is either P or C=In ; if Action is O=Out . Note that the buffer <i>must</i> be initialized with spaces before Open .

Calling the Create Window Key API from VB

The Create Window Key function can be used to create the **window keys** used in matching. They are constructed from elements in the input record, which is then used to select records for inclusion to the Match window for comparison. *For detailed information about Create Window Key functions, see the Batch User's Guide.*

The following Declare statement is to call the Create Window Key **cf_crwkey** function from a VB application:

```
Declare Sub Winkey Lib "cflibstd.dll" Alias "_cf_crwkey@20" _  
    (ByVal Action As String, _  
     ByVal ParmFile As String, _  
     ByVal Record As String, _  
     ByVal WindowKey As String, _  
     ByVal ControlArea As String)
```

This table describes parameters used for VB calls to the Create Window Key function.

Table 3.42 Create Window Key (cf_crwkey) Parameters (VB)

Parameter	Type	Use	Length	Description
Action	String	In	1	Defines call type as O=Open , P=Process , or C=Close .
ParmFile	String	In	variable	Name of the parameter file.
Record	String	In	variable	Record that contains the data used to derive the window key.
WindowKey	String	Out	1500	Buffer that holds up to thirty 50-byte window keys. Note that the buffer <i>must</i> be initialized with spaces before Open . Keys are left-justified every 50 bytes.
ControlArea	String	In/ Out	4	A 4-byte pointer used as a control work area. Note that the buffer <i>must</i> be initialized with spaces before Open . If Action is either P or C=In ; if Action is O=Out .

Calling the Matcher API from VB

The Matcher differs from the other modules because many calls to individual functions occur within the Matcher module itself. The match type defines which set of functions is required. There are two matching modes, **Reference Matching** and **Window Matching**; these matching modes are described in the following sections. For detailed information about Matcher functions, see the *Batch User's Guide*.

Reference Matching

This section discusses the steps required to use the Matcher in the Reference Match mode. This mode of operation matches one record, called the *candidate record*, to one or more records, called the *reference record*, that have been retrieved from a source such as a database. The match rules are contained in

external parameter files that are loaded to memory upon initialization. The driver program *must* link to the library of Matcher routines.

Reference Match Requirements

- Candidate record and reference record(s) *must* both have the same shape.
- Reference record(s) *must* be processed using the same or similar steps as the candidate record to ensure that the all records use standardized fields.
- Matcher Field List parameter files *must* be configured to use field locations (**fldlocs**) not DLL field names, as is commonly used in batch.
- If an optional candidate matching information key is used, reference records *must* contain unique keys for both household and individual levels.

Reference Match Step Names

This table lists step names that can be used as an example of the minimum steps required to perform a reference match and retrieve the results. The step name is given followed by the entry point name in parenthesis in the program. Each module section of the documentation details its function.

- ① *Required steps that are external to the project, but required for the match process, are not included below. For example, a step which extracts records from a database to load to the Match window is an operation that is performed by either the calling program or another external subroutine.*

Table 3.43 Reference Match Step Names (VB)

Function Name	Description
m_initmr	Call Initialize Matcher to set up the Match windows and initialize the Matcher.

Table 3.43 Reference Match Step Names (VB) (Continued)

Function Name	Description
m_params	Call Process Parameters four times for a retail match; twice for a commercial match. Each call <i>must</i> process one of the required fields or patterns parameter files required for the Matcher. The field files <i>must</i> use the fidlocs keyword instead of fidnames , because DDLs are not used in the call process.
m_addrec	Call Add Record to load the Match window with reference records. For the number of records with window key of the same value as the candidate record.
m_cmatch	Call Match Candidate to match the transaction record to the database records that have been loaded to the Match window during Add Record .
m_nummat	Call Number Reference Matches to obtain the number of records that matched to the candidate record.
m_getmat	Call Obtain Reference Match to return a value (in Number of Reference Matches) that is equal to the total number of reference match records that are available to be obtained. (One record is called at a time.) <ul style="list-style-type: none"> • Each record is written to a buffer. It must be moved from the buffer before calling for another record. Otherwise, the original record is overwritten. • Buffer is the input record length plus 3 bytes holding the match pattern ID number (e.g. P150 would have a pattern ID of 150). • By default, the returned buffer is sorted in match pattern ID ascending order. If this order is not desired, pattern ID numbers can be set to the same value. • Calling parameters specify if you are requesting a household/commercial match or an individual match.
m_clearw	Call Clear Window to empty the Match window, then restart with Add Record to match the next transaction.
m_endmat	Call End Matcher after all matching is complete to close the Matcher and write statistics to a file.

① *You can retrieve Suspect Matches in a similar way if you are using suspect match patterns. Call **Number of Reference Suspects (m_numsus)** and use the value returned to loop for calling **Obtain***

Reference Suspects (m_getsus).

Window Matching

This section discusses the steps required to use the Matcher in the Window Match mode. This mode of operation matches a group of records to one another. The Match rules are contained in external parameter files that are loaded to memory upon initialization. The driver program *must* link to the library of Matcher routines.

Window Match Step Names and Module Names

The following list can be used as an example of the minimum steps required to perform a window Match and retrieve the results. Each module section of the documentation details its function. **Steps external to the Trillium Software process, but necessary for the Match process are not in this list.**

Table 3.44 Window Match Function Names (VB)

Function	Description
m_initmr	Call Initialize Matcher to set up the Match windows and initialize the Matcher.
m_params	Call Process Parameters four times for a Retail match; twice for a Commercial match. Each call <i>must</i> process one of the required fields or patterns parameter files required for the Matcher. Field files <i>must</i> use the fldlocs keyword instead of the fldnames keyword because DDLs are not used in the API process.
m_addrec	Call Add Record For the number of records with the same candidate code to add to a Match window.
m_wmatch	Call Match Window to match the records that have been loaded to the Match window during Add Record .
m_common	Call Create Common to append the common data and survivor flag information on the matched records. <i>Window match only. Optional.</i>
m_retrvr	Call Retrieve Records for the number of window records to retrieve the matched record with appended match information. Return buffer <i>must</i> equal input record length + 184 bytes.

Table 3.44 Window Match Function Names (VB) (Continued)

Function	Description
m_clearw	Call Clear Window to empty the Match window and restart with Add Record .
m_endmat	Call End Matcher after all matching is complete to close the Matcher and write statistics.

Add Record (m_addrec)

The **Add Record (m_addrec)** function loads the Match window with records.

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_addrec** function:

```
Declare Sub AddRecord Lib "mlibstd.dll" Alias "_m_addrec@16" _
    (ByVal RetCode As String, _
     ByVal WindowType As String, _
     ByVal CandRec As String, _
     ByRef ControlArea As Variant)
```

This table describes parameters used for VB calls to the **m_addrec** function.

**Table 3.45 Add Record
(m_addrec) Parameters (VB)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. Buffer <i>must</i> be initialized with spaces. See the section "Error Codes from Add Records" in Chapter 4 of Volume II.
WindowType	String	In	1	Indicates which window to expand, either R=Retail or C=Commercial .
CandRec	String	In	variable	The record to add.
ControlArea	Variant	In	16	Control work area used by Matcher.

Clear Window (m_clearw)

The **Clear Window (m_clearw)** function erases the specified window record list and produces window statistics.

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_clearw** function:

```
Declare Sub ClearWindow Lib "mlibstd.dll" Alias "_m_clearw@12" _
    (ByVal RetCode As String, _
     ByVal WindowType As String, _
     ByRef ControlArea As Variant)
```

This table describes parameters used for VB calls to the **m_clearw** function.

**Table 3.46 Clear Window
(m_clearw) Parameters (VB)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. <i>See the section "Error Codes from Clear Window" in Chapter 4 of Volume II.</i>
WindowType	String	In	1	Indicates the window to erase; either R=Retail or C=Commercial .
ControlArea	Variant	In	16	Control work area used by the Matcher.

Create Common (m_common)

The **Create Common** function (*Window matching only*) creates the common data segment for matched records in a record list. This function also produces the linked records file, if desired. Common data is built of the "best" data from matches, based on the rules defined in the common fields parameter file.

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_common** function:

```
Declare Sub CreateCommon Lib "mllibstd.dll" Alias "_m_common@12" _  
    (ByVal RetCode As String, _  
     ByVal WindowType As String, _  
     ByRef ControlArea As Variant)
```

This table describes parameters used for VB calls to the **m_common** function.

**Table 3.47 Create Common
(m_common) Parameters (VB)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See the section "Error Codes from Create Common" in Chapter 4 of Volume II.
WindowType	String	In	1	Indicates the window from which the common data segment is generated; R=Retail or C=Commercial .
ControlArea	Variant	In	16	Control work area used by the Matcher.

End Matcher (m_endmat)

The **End Matcher (m_endmat)** function displays program statistics, erases both Retail and Commercial window record lists, if they still exist, and terminates the Matcher.

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_endmat** function:

```
Declare Sub EndMatcher Lib "mlibstd.dll" Alias "_m_endmat@8" _
    (ByVal RetCode As String, _
     ByRef ControlArea As Variant)
```

This table describes the parameters used by VB calls to the **m_endmat** function.

**Table 3.48 End Matcher
(m_endmat) Parameters (VB)**

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See the section "Error Codes from End Matcher" in Chapter 4 of Volume II.
ControlArea	Variant	In	16	Control work area used by the Matcher.

Initialize Matcher (m_initmr)

The **Initialize Matcher (m_initmr)** function creates the Match window areas for either a Reference match or a Window match.

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_initmr** function:

```
Declare Sub InitMatcher Lib "mlibstd.dll" Alias "_m_initmr@20" _
    (ByVal RetCode As String, _
     ByVal StatFile As String, _
     ByVal LinkFile As String, _
     ByRef RecLen As Long, _
     ByRef ControlArea As Variant)
```

This table describes the parameters used by VB calls to the **m_initmr** function.

Table 3.49 Initialize Matcher (m_initmr) Parameters (VB)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See the section "Error Codes from Initialize Matcher" in Chapter 4 of Volume II.
StatFile	String	In	variable	Name of the file that contains matching statistics.
LinkFile	String	In	variable	Name of the file to contain keys of matched records when performing multiple window match scenarios. This file is only required when performing window matching.
RecLen	Long	In	variable	Length of the records to match. A negative length activates the Matcher tracing.
ControlArea	Variant	Out	16	Control work area used by the Matcher. Buffer mst be initialized with spaces.

Match Candidate (m_cmatch)

The **Match Candidate (m_cmatch)** function initiates the matching of a specified candidate record against records in the specified window record list, and saves information about the best matches. (*Reference match only*)

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_cmatch** function:

```
Declare Sub MatchCand Lib "mlibstd.dll" Alias "_m_cmatch@16" _
    (ByVal RetCode As String, _
     ByVal WindowType As String, _
     ByVal CandRec As String, _
     ByRef ControlArea As Variant)
```

This table describes parameters used for VB calls to the **m_cmatch** function.

Table 3.50 Match Candidate (m_cmatch) Parameters (VB)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Match return code. See "Error Codes from Match Candidate" in Chapter 4 of Volume II.
WindowType	String	In	1	Indicates the window to use for the match, and therefore, the type: R =Retail C =Commercial
CandRec	String	In	variable	Candidate record to match.
ControlArea	Variant	In	16	Control work area used by the Matcher.

Match Window (m_wmatch)

The **Match Window (m_wmatch)** function initiates the matching of a specified window record list against itself. The **m_wmatch** function then appends the results of the matching process to each record in the record list (*Window match only*).

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_wmatch** function:

```
Declare Sub MatchWindow Lib "mllibstd.dll" Alias "_m_wmatch@12" _  
    (ByVal RetCode As String, _  
     ByVal WindowType As String, _  
     ByRef ControlArea As Variant)
```

This table describes the parameters used to call the **m_wmatch** function from VB applications.

Table 3.51 Match Window (m_wmatch) Parameters (VB)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See "Error Codes from Match Window" in Chapter 4 of Volume II.
WindowType	String	In	1	Indicates the window to use in the match, and therefore, the type of matching: R =Retail C =Commercial
ControlArea	Variant	In	16	Control work area used by Matcher.

Number of Reference Matches (m_nummat)

The Number of Reference Matches (**m_nummat**) function finds the number of matches that occurred on a Reference match.

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_nummat** function:

```
Declare Sub NumRefMats Lib "mlibstd.dll" Alias "_m_nummat@16" _
    (ByVal RetCode As String, _
     ByVal MatchLevel As String, _
     ByRef NumMats As Long, _
     ByRef ControlArea As Variant)
```

This table describes parameters used for VB calls to the **m_nummat** function.

Table 3.52 Number of Reference Matches (m_nummat) Parameters (VB)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See the section "Error Codes from Number of Reference Matches" in Chapter 4 of Volume II.
MatchLevel	String	In	1	Match level: 0 =Household/business 1 =Individual
NumMats	Long	Out	variable	Number of matches.
ControlArea	Variant	In	16	Control work area used by Matcher.

Number of Reference Suspects (m_numsus)

The **Number of Reference Suspects (m_numsus)** function finds the number of suspects matches which occurred on a Reference match.

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_numsus** function:

```
Declare Sub NumRefSusps Lib "mlibstd.dll" Alias "_m_numsus@16" _
    (ByVal RetCode As String, _
     ByVal MatchLevel As String, _
     ByRef NumSusps As Long, _
     ByRef ControlArea As Variant)
```

This table describes the parameters used by VB to call the **m_numsus** function.

Table 3.53 Number of Reference Suspects (m_numsus) Parameters (VB)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See the section "Error Codes from Number of Reference Suspects" in Chapter 4 of Volume II.
MatchLevel	String	In	1	Match level: 0=Household/business , or 1=Individual .
NumSusps	Long	Out	variable	Number of suspects.
ControlArea	Variant	In	16	Control work area used by Matcher.

Obtain References Matches (m_getmat)

The **Obtain Reference Matches (m_getmat)** function obtains the specified record that matched successfully on a Reference match from the specified window.

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_getmat** function:

```
Declare Sub ObtainRefMat Lib "mllibstd.dll" Alias "_m_getmat@20" _
    (ByVal RetCode As String, _
     ByVal MatchLevel As String, _
     ByRef WhichMatch As Long, _
     ByVal Buffer As String, _
     ByRef ControlArea As Variant)
```

This table describes parameters used by VB calls to the **m_getmat** function.

Table 3.54 Obtain Reference Matches (m_getmat) Parameters (VB)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. <i>See the section "Error Codes from Obtain Reference Matches" in Chapter 4 of Volume II.</i>
MatchLevel	String	In	1	Match level, either 0=Household/business , or 1=Individual .
WhichMatch	Long	In	n ¹	Match to retrieve.
Buffer	String	Out	n ²	Contains the retrieved record.
ControlArea	Variant	In	16	Control work area used by Matcher.

n¹ = Length of numeric value indicating which match to retrieve.

n² = Length of the record as defined during the initializeMatcher call + 3 bytes to contain the matched pattern ID.

Obtain Reference Suspects (m_getsus)

The **Obtain Reference Suspects (m_getsus)** function obtains the specified record that suspected successfully on a Reference match from the specified window. The following Declare statement is required in order to use Visual Basic (VB) to call the **m_getsus** function:

```
Declare Sub ObtainRefSusp Lib "mlibstd.dll" Alias "_m_getsus@20" _
    (ByVal RetCode As String, _
     ByVal MatchLevel As String, _
     ByRef WhichSuspect As Long, _
     ByVal Buffer As String, _
     ByRef ControlArea As Variant)
```

This table lists parameters used by VB calls to the **m_getsus** function.

Table 3.55 Obtain Reference Suspects (m_getsus) Parameters (VB)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See the section "Error Codes from Obtain Reference Suspects" in Chapter 4 of Volume II.
MatchLevel	String	In	1	Match level. Set to either: 0=Household/business , or 1=Individual .
WhichSuspect	Long	In	n ¹	Contains which suspect match to retrieve from match window.
Buffer	String	Out	n ²	Contains the retrieved record.
ControlArea	Variant	In	16	Control work area used by Matcher.

n¹ = Length of numeric value indicating which suspect match to retrieve.

n² = Length of the record as defined during the initializeMatcher call + 3 bytes to contain the suspect pattern ID.

Process Parameters (m_params)

The **Process Parameters (m_params)** function processes a Matcher parameter file (either a field list or a pattern list).

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_params** function:

```
Declare Sub ProcessParms Lib "mlibstd.dll" Alias "_m_params@24" _
    (ByVal RetCode As String, _
     ByVal ParmFName As String, _
     ByVal ParmType As String, _
     ByVal Handle As String, _
     ByVal Rname As String, _
     ByRef ControlArea As Variant)
```

This table describes the parameters used by VB calls to the **m_params** function.

Table 3.56 Process Parameters (m_params) Parameters (VB)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. <i>See the section "Error Codes from Process Parameters" in Chapter 4 of Volume II.</i>
ParmFName	String	In	variable	Name of parameter file to process.
ParmType	String	In	1	Type of parameter file to process. <i>See the next table for these values.</i>
Handle	String	-	-	Reserved for future use. Pass nulls.
Rname	String	-	-	Reserved for future use. Pass nulls.
ControlArea	Variant	In	16	Control work area used by Matcher.

The possible values for the **ParmType** parameter are defined below.

Table 3.57 ParmType Parameter Values (VB)

Value	Description
0	Field list parameters for producing Common Data Segment: <ul style="list-style-type: none">• branch_number• middlename• business_name• lastname• first_name• postalcode• gender• street_title• house_number• source_id <p><i>① When the specified 'parmtime' value is "0", the following field descriptions, along with dictionary field names or field positions and length values, should appear in the specified parameter file to allow for the creation of the common data segment.</i></p>
1	Household field comparison routine list.
2	Individual field comparison routine list.
3	Business field comparison routine list.
4	Household grade pattern list.
5	Individual grade pattern list.
6	Business grade pattern list.

Retrieve Record (m_retrvr)

The **Retrieve Record (m_retrvr)** function retrieves the specified relative record from a specified window record list.

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_retrvr** function:

```
Declare Sub RetrieveRecord Lib "mlibstd.dll" Alias "_m_retrvr@20" _
    (ByVal RetCode As String, _
     ByVal WindowType As String, _
     ByRef RecNum As Long, _
     ByVal WinRec As String, _
     ByRef ControlArea As Variant)
```

This table describes parameters used by VB calls to the **m_retrvr** function.

Table 3.58 Retrieve Record (m_retrvr) Parameters (VB)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Return code. See "Error Codes from Retrieve Records" in Chapter 4 of Volume II.
WindowType	String	In	1	Indicates which window to use; either R=Retail or C=Commercial .
RecNum	Long	In	n ¹	Indicates the record to retrieve.
WinRec	String	Out	n ²	Contains the retrieved record.
ControlArea	Variant	In	16	Control work area used by Matcher.

n¹ = Length of numeric value indicating which match to retrieve.

n² = Length of the record as defined during the initializeMatcher call + 184 bytes to contain the window match key results.

Sort Window (m_sortwn)

The **Sort Window (m_sortwn)** function sorts the matched window records into a specified order (for example, matched individuals in matched household).

The following Declare statement is required in order to use Visual Basic (VB) to call the **m_sortwn** function:

```
Declare Sub SortWindow Lib "mllibstd.dll" Alias "_m_sortwn@16" _  
    (ByVal RetCode As String, _  
     ByVal WindowType As String, _  
     ByVal SortOrder As String, _  
     ByRef ControlArea As Variant)
```

This table describes the parameters used by VB to call the **m_sortw** function.

Table 3.59 Sort Window (m_sortw) Parameters (VB)

Parameter	Type	Use	Length	Description
RetCode	String	Out	1	Sort return codes. See the section "Error Codes from Sort Window" in Chapter 4 of Volume II.
WindowType	String	In	1	Indicates the type of Matching window to use in the match; either R=Retail or C=Commercial .
SortOrder	String	In	1	Indicates keys used for the sort; either: –Matched individuals within matched households –Suspect individuals within suspect households –Matched individuals within suspect individuals within matched households
ControlArea	Variant	In	16	Control work area used by the Matcher

CHAPTER 4

Trillium Software System API Calls on UNIX

This chapter describes the API calls to the Trillium Software System® on the UNIX platform. It includes information about the following calling languages:

- C
- C++
- Java

This chapter also describes how to call the XML API from C.

❗ *Paths set within a parameter file cannot exceed 80 characters.*

API Calls from C on UNIX

This section describes C language calls to the Trillium Software System® Applications Programming Interface (API) on UNIX. See the *Batch User's Guide* for complete information about the functions of each individual module.

- To *statically* link to the Trillium Software System® libraries, use the libraries located in the `.\lib` directory of the Trillium Software System base installation.
- To *dynamically* link to the Trillium Software System® libraries, use the libraries located in the `.\cdll` directory of the Trillium Software System base installation.

Sample programs of calling the modules are located in the `.\samples` directory in the same area.

- ① *'C' calls to each module must include the respective header file, as described in the following table and each corresponding section.*

Include this header file	For 'C' Calls to this module
gaclient.h	Business Data Parser
cfclient.h	Converter
maclient.h	Matcher

'C' Call Types for the 'Action' Parameter

There are three different call types as detailed by the **Action** parameter. Each call uses the same parameters with a different **Action** value. The **Action** parameter tells the program what function to perform for a specific call. The settings for the **Action** parameter are described in the following table.

- O Open** Initializes the module, loads required tables into memory, and reads the parameter (parm) file. For the Geocoder, it Creates the control area, opens Level 1 index into memory and sets up control areas from parameters. Also opens Level 2 and BaseData files and saves the file handles in the control area.

- P Process** Processes data in the Input buffer and writes data to the Output buffers. For the Geocoder, Uses control areas and its variables; all other variables are automatic (unique copies to each function). Reads Level 2 and Basedata files as needed and applies matching logic. Also reads the memory copy of Level 1.
- R Minimal Open** For the *Postal Geocoders only*, functions the same as Process, except uses an automatic version of the Level 2 and Basedata file handles. (Performs its own file open/close on these two resource files.) **REQUIRED** for multi-threaded use of Geocoder.
- C Close** Closes the modules, cleans up allocated resources, and writes any statistical information.

Calling the Converter API from C

cf_coni Function

The **cf_coni** function is used to call the Converter.

Syntax

```
cf_coni(Action, Control, ParmFile, ParmEcho, Input, Output, RetCode);
```

The **cf_coni** function parameters are described in Table 4.1.

Example

```
cf_coni("O",
        &control,
        parmfile,
        parmecho,
        inarea,
        outarea,
        &retcode)
```

Table 4.1, “Converter (cf_coni) Parameters (C on UNIX)” describes parameters used by the **cf_coni** function to call the Converter.

Table 4.1 Converter (cf_coni) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
Action	Char	In	1	Defines the call type as either O=Open , P=Process or C=Close
Control	Void **	In/ Out	16	Control work area created on the Open call, then used during Process and Close calls. If Action is either P or C=In ; if O=Out .
parmFile	Char	In	n*	Name of parameter file.
ParmEcho	Char	Out	n*	Name of parameter echo file used for debugging. If not used, set to NULL. <i>Optional</i> .
Input	Char	In	Variable*	Buffer that contains input data to process. Note that length should equal the record length specified in the input DDL.

Table 4.1 Converter (cf_coni) Parameters (C on UNIX) (Continued)

Parameter	Type	Use	Length	Description
Output	Char	Out	Variable*	Buffer containing output data. Note that length should equal the record length specified in the output DDL.
RetCode	Char	Out	1	Return code value that indicates the status of the program call. See the section "Converter Error Codes" in Chapter 4 of Volume II.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

Calling the Business Data Parser API from C

ga_prsi Function

The **ga_prsi** function calls the Business Data Parser. This function uses Action variations in the calling program to open the program, parse input data, and close the program. A *Parser Service call* can also be used to create an error message in the event of a program problem, and can also be used to display data during processing which can be used for debugging format problems.

Syntax

ga_prsi(Action, Parmarea, Input, Output, NULL, ParmFile, Control);

Example

```
ga_prsi("O",
        parmarea,
        inarea,
        outarea,
        NULL,
        parmfile,
        &control[0])
```

Table 4.2, "Business Data Parser (ga_prsi) Parameters (C on UNIX)" describes parameters listed in the preceding sample call statement.

Table 4.2 Business Data Parser (ga_prsi) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
Action	Char	In	1	Defines call type as O=Open, P=Process, or C=Close.
ParmArea	Char	Out	10,000	Contains Business Data Parser parameter data.
Input	Char	In	1,000	Input buffer that contains the input data to be processed when the Process call is used.
Output	Char	Out	13,000	Buffer containing output data. <i>See the Business Data Parser PREPOS section in Chapter 4 of Volume II.</i>
NULL	-	-	4	A 4-byte pointer filled with null values (x'00').
ParmFile	Char	In	n*	Name of parameter file.
Control	Void **	In/ Out	16	Control work area created on the Open call, then used during the Process and Close calls. If Action is either P or C=In ; or if Action is O=Out .

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

Calling the Create Window Key API from C

A *window key* is composed from elements in the input record and is used to select records for inclusion to the Match window for comparison. The Create Window Key program uses a rules-based parameter file to create up to 30 window keys per record. This program requires a parameter file and a data dictionary file (DDL). The parameter file will define the DDL file name and the rules file name that details the window key construction methods.

For more information about Create Window Key functionality, see the *Batch User's Guide*.

cf_crwkey Function

The **cf_crwkey** function is used to call the **Create Window Key** program. This function uses the following syntax:

Syntax

```
cf_crwkey(Action, Parmfile, Curr_record, WindowKey, Control);
```

Example

```
cf_crwkey("O",  
          "parmfile",  
          curr_record,  
          windowkey,  
          &control);
```

Create Window Key Parameter Files

Only two parameter files are required to call the **Create Window Key** function (the Batch version requires four):

```
INP_DDL      ../dict/parsgout.ddl,PARSGOUT  
RULES_DDNAME ../parms/pfrules
```

Table 4.3, “cf_crwkey Parameters (C on UNIX)” describes parameters used by C calls to the Create Window Key program.

Table 4.3 cf_crwkey Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
Action	Char	In	1	Defines the call type as either O=Open , P=Process , or C=Close .
ParmFile	Char	In	n*	Name of parameter file.
Curr_Record	Char	In	n*	Record containing data used to derive the window key.
WindowKey	Char	Out	1500	Buffer to hold window key results; can contain up to thirty 50-byte window keys. Keys are left-justified every 50 bytes.
Control	Void **	In/ Out	4	Control work area built by the program on the Open call, then used during the Process and Close calls. If Action is either P or C=In ; or if Action is O=Out .

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

Calling the Matcher API from C

The Matcher has two modes: Reference matching and Window matching. The Matcher differs from other modules because many calls to individual functions within the Matcher module itself are called. The type of match to perform dictates which functions you need to call. The two matching modes are *Reference matching* and *Window matching*. For complete information about Matcher functionality, see the *Batch User's Guide*.

Reference Matching Mode

Reference Matching mode matches one record (called the candidate record) to one or more records (called the reference record) that have been retrieved from a source such as a database. The match rules are contained in external parameter files that are loaded to memory upon initialization. The following requirements *must* be met for a Reference match:

- Both candidate and reference record(s) must be the same shape.
- Reference records must have been processed through the same or similar steps as the candidate record to ensure field standardization between the records.
- Matcher fields list parameter files must be configured to use field locations (*fldlocs*) and not DDL field names, as is commonly used in batch processing.

Reference Match Step and Module Names

The following list can be used as an example of the *minimum* steps required to perform a Reference match and retrieve the results. The step name is given followed by the entry point name in parenthesis in the program. Each module section of the documentation details its function.

Steps external to the project, but necessary to the match process are not in this list; for example, pulling records from a database to load to the Match window is an operation that is performed by either the calling program or another external subroutine.

Table 4.4, "Suggested Reference Match Function Name Order (C on UNIX)" describes this order.

Table 4.4 Suggested Reference Match Function Name Order (C on UNIX)

Function	Description
m_initmr	Call Initialize Matcher to set up the Match windows and initialize the Matcher.
m_params	Call Process Parameters four times for a retail match; twice for commercial match. Each call <i>must</i> process one of the required fields or patterns parameter files required for the matcher. Field files <i>must</i> use the fldlocs keyword instead of the fldnames keyword because DDLs are not used in the calling process.
m_addrrec	Call Add Record to load the Match window with reference records. (Records with a window key of the same value as the candidate record.)
m_cmatch	Call Match Candidate to match transaction record to the database records that were loaded to the Match window during Add Record .

Table 4.4 Suggested Reference Match Function Name Order (C on UNIX)

Function	Description
<code>m_nummat</code>	Call Number of Reference Matches to obtain the number of records that matched to the candidate record.
<code>m_getmat</code>	Call Obtain Reference Match to return a value in Number of Reference Matches that is equal to the total number of reference match records that are available to be obtained. (One record is called at a time.) <ul style="list-style-type: none"> • Each individual record is written to a buffer. That record must be moved from the buffer before calling for another record. Otherwise, original record is overwritten. • Buffer is the input record length plus 3 bytes holding the match pattern ID number (e.g. P150 would have a pattern ID of 150). • By default, the returned buffer is sorted in match pattern ID ascending order. If this order is not desired, the pattern ID numbers can all be set to the same value. • Calling parameters specify if you are requesting a household/commercial match or an individual match.
<code>m_clearw</code>	Call Clear Window to empty the Match window, then restart with Add Record .
<code>m_endmat</code>	Call End Matcher after all matching is complete to close and write statistics to a file.

① *Suspect matches can be retrieved in a similar way if suspect match patterns are being used. Call the **Number of Reference Suspects** (`m_numsus`) function and use the value returned to loop for calling the **Obtain Reference Suspects** (`m_getsus`) function.*

Window Matching Mode

Window Matching matches a group of records to one another. The match rules are contained in parameter files that are loaded to memory upon initialization.

Window Match Step and Module Names

Table 4.5, “Suggested Window Match Name Order (C on UNIX)” shows an example of the *minimum* steps required to perform a Window match and retrieve the results. The step name is given followed by the entry point name in parenthesis in the program. Each module section of the documentation details its function.

Table 4.5 Suggested Window Match Name Order (C on UNIX)

Function	Description
m_initmr	Call Initialize Matcher to set up the Match windows and initialize the Matcher.
m_params	Call Process Parameters four times for a <i>retail</i> match; twice for a <i>commercial</i> match. <ul style="list-style-type: none"> • Each call <i>must</i> process one of the required fields or patterns parameter files required for the Matcher. • Field files <i>must</i> use the fldlocs keyword because DDLs are not used in the calling process.
m_addrac	Call Add Record for the number of records with window key of the same value as the candidate record to load the Match window with reference records.
m_wmatch	Call Match Window to match the records that have been loaded to the Match window during Add Record .
m_common	Call Create Common to optionally append the common data and survivor flag information on the matched records (<i>Window match only</i>).
m_retrvr	Call Retrieve Relative Record for the number of window records to retrieve the matched record with appended match information. Buffer is always the input record length + 184 bytes.
m_sortwn	Call Sort Window to sort the matched window records into a specified order (such as matched individuals in matched household). <i>Optional</i> .
m_clearw	Call Clear Window to empty the Match window then restart with Add Record.
m_endmat	Call End Matcher after all matching is complete to close and write statistics to a file.

m_addréc Function

The **m_addréc** function loads the Match window with records. This function uses the following syntax:

Syntax

```
Add_Record(RetCode, MA_Retail_Flag, Curr_Record, UserArea);
```

Example

```
Add_Record(&ret,
            MA_Retail_Flag,
            curr_record,
            userarea);
```

Table 4.6, “Add Records (m_addréc) Parameters (C on UNIX)” describes the parameters used by the m_addréc function.

Table 4.6 Add Records (m_addréc) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned by the subroutine. Check this value after calling. See “Error Codes from Add Records” in Chapter 4 of Volume II.
MA_Retail_Flag	Char	In	1	Indicates the window to add the record; either R=Retail , or C=Commercial .
Curr_Record	Char	In	n*	Record to add. Length <i>must</i> equal record length defined in Initialize Matcher call.
UserArea	Void **	In	16	Address of parameter area that resides in the calling program and is maintained by the Matcher. Area is created during Initialize Matcher , passed back to the calling program, and <i>must</i> be passed to each subsequent call.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

m_clearw Function

The **m_clearw** function erases the specified window record list and can optionally produce window statistics. This function uses the following syntax:

Syntax

```
m_clearw(ret, MA_Retail_Flag, userarea);
```

Example

```
m_clearw(&ret,
        MA_Retail_Flag,
        userarea);
```

Table 4.7, “Clear Window (m_clearw) Parameters (C on UNIX)” describes these parameters.

Table 4.7 Clear Window (m_clearw) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned. <i>See the section “Error Codes from Clear Window” in Chapter 4 of Volume II.</i>
MA_Retail_Flag	Char	In	1	Indicates the window to add record; either R=Retail , or C=Commercial .
UserArea	Void **	In	16	Address of parameter area that resides in the API program and maintained by the Matcher. Created during Initialize Matcher , passed back to the API program, and <i>must</i> be passed to each subsequent call.

m_common Function

The **m_common** function creates the *common data segment* for matched records in a record list. Also, produces the linked records file, if desired. The common data is built of the "best" data from matches based on the rules defined in the common fields parameter file. (*Used for Window Matching only.*)

Syntax

```
m_common(RetCode, MA_Retail_Flag, UserArea);
```

Example

```
m_common(&ret,
         MA_Retail_Flag,
         userarea);
```

This table describes parameters used by C calls to the **m_common** function.

**Table 4.8 Create Common
(m_common) Parameters (C on UNIX)**

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned by the subroutine. Check this value after calling. See the "Error Codes from Create Common" section in Chapter 4 of Volume II.
MA_Retail_Flag	Char	In	1	Indicates the window to generate the common data segment; either R=Retail , or C=Commercial .
UserArea	Void **	In	16	Address of parameter area that resides in calling program and maintained by the Matcher. Area created during Initialize Matcher , passed back to the calling program, and <i>must</i> be passed to each subsequent call.

m_endmat Function

The **m_endmat** function displays the program statistics, if desired. Erases both the retail and commercial window record lists, if they still exist, and terminates the Matcher.

Syntax

m_endmat(RetCode, UserArea)

Example

```
m_endmat (&ret,
          userarea)
```

Table 4.9, “End Matcher (m_endmat) Parameters (C on UNIX)” describes parameters used for C calls to the **m_endmat** function.

**Table 4.9 End Matcher
(m_endmat) Parameters (C on UNIX)**

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned by the subroutine. <i>See the "Error Codes from End Matcher" section in Chapter 4 of Volume II.</i>
UserArea	Char	In	16	Address of parameter area that resides in the calling program and maintained by the Matcher. Area is created during Initialize Matcher , passed back to the calling program, and <i>must</i> be passed to each subsequent call.

m_initmr Function

The **m_initmr** function initializes the Matcher by creating the Match window areas for either a Reference match or a Window match.

Syntax

```
m_initmr(&ret, stat_name, link_name, &match_lrecl, userarea);
```

Example

```
m_initmr(&ret,
         stat_name,
         link_name,
         &match_lrecl,
         userarea);
```

Table 4.10, “Initialize Match Window (m_initmr) Parameters (C on UNIX)” describes parameters used by C calls to Initialize Matcher function:

Table 4.10 Initialize Match Window (m_initmr) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned by the subroutine. See the “Error Codes from Initialize Matcher” section in Chapter 4 of Volume II.
Stat_Name	Char	In	n*	Name of file to write match statistics.
Link_Name	Char	In	n*	Name of the file to contain keys of matched records when performing multiple window match scenarios. REQUIRED only to perform window matching.

Table 4.10 Initialize Match Window (m_initmr) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
Match_Lrecl	Int ptr	In		Length of the input records to match. A negative record length activates a debug option that writes data to the statistics file.
UserArea	Void **	Out	16	Address of parameter area that resides in the calling program; maintained by the Matcher. Area is created during Initialize Matcher , passed back to the calling program, and <i>must</i> be passed to each subsequent call.

m_cmatch Function

The **m_cmatch** function initiates matching of a specified candidate record against records in the specified window record list and saves information about the best matches (*Reference match*).

Syntax

```
m_cmatch(&ret, MA_Retail_Flag, input, userarea);
```

Example

```
m_cmatch(&ret,
         MA_Retail_Flag,
         input,
         userarea);
```

Table 4.11, “Match Candidate (m_cmatch) Parameters (C on UNIX)” describes parameters used by C calls to the **m_cmatch** function.

Table 4.11 Match Candidate (m_cmatch) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned by the subroutine. <i>See the “Error Codes from Match Candidate” section in Chapter 4 of Volume II.</i>
MA_Retail_Flag	Char	In	1	Indicates the window to match Candidate record; either R=Retail , or C=Commercial .
Input	Char	In	n*	Buffer that contains candidate record to match to the records in the window.
UserArea	Void **	In	16	Address of parameter area residing in the calling program and maintained by the Matcher. Created during Initialize Matcher , passed back to the calling program, and <i>must</i> be passed to each subsequent call.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

m_wmatch Function

The **m_wmatch** function initiates the matching of a specified window record list against itself. Appends the results of the matching process to each record in the record list (*Window match*).

Syntax

```
m_wmatch(&ret, MA_Retail_Flag, userarea);
```

Example

```
m_wmatch(&ret,
         MA_Retail_Flag,
         userarea);
```

Table 4.12, “Match Window (m_wmatch) Parameters (C on UNIX)” describes parameters used for C calls to the **m_wmatch** function.

Table 4.12 Match Window (m_wmatch) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned by the subroutine. See the “Error Codes from Match Window” section in Chapter 4 of Volume II.
MA_Retail_Flag	Char	In	1	Indicates the window to start matching: R=Retail; C=Commercial
Userarea	Void **	In	16	Address of parameter area that resides in the calling program and maintained by the Matcher. Area is created during Initialize Matcher , passed back to the calling program, and <i>must</i> be passed to each subsequent call.

m_nummat Function

The **m_nummat** function finds the number of matches that occurred on a reference match.

Syntax

```
m_nummat(&ret, &level, &num_matches, userarea);
```

Example

```
m_nummat(&ret,
         &level,
         &num_matches,
         userarea);
```

Table 4.13, “Number Reference Matches (m_nummat) Parameters (C on UNIX)” describes parameters used by C calls to the **m_nummat** function.

**Table 4.13 Number Reference Matches
(m_nummat) Parameters (C on UNIX)**

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned. <i>See the “Error Codes from Number Reference Matches” section in Chapter 4 of Volume II.</i>
Level	Char	In	1	Indicator for match level; either 0=Household or 1=individual . A <i>Commercial</i> match is the same as a <i>Household</i> match, and therefore uses a 0 value.
Num_Matches	Long Int	Out	n*	Resulting number of matches in the window.
Userarea	Void **	In	16	Address of the parameter area residing in the calling program and maintained by the matcher. This area is created during Initialize Matcher , passed back to the calling program, and <i>must</i> be passed to each subsequent call.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

m_numsus Function

The **m_numsus** function finds the number of suspects matches which occurred on a reference match.

Syntax

```
m_numsus(&ret, &level, &num suspects, userarea);
```

Example

```
m_numsus (&ret,
          &level,
          &num suspects,
          userarea);
```

Table 4.14, “Number of Suspect Matches (m_numsus) Parameters (C on UNIX)” describes parameters used by C calls to the **m_numsus** function.

Table 4.14 Number of Suspect Matches (m_numsus) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned by the subroutine.
Level	Char	In	1	Match level; either 0 = Household or 1 = Individual . A <i>Commercial</i> match is the same as <i>Household</i> , and uses a 0 value.
Num_Suspects	Char	Out	n*	Resulting number of suspect matches.
UserArea	Void **	In	16	Address of parameter area residing in the calling program and maintained by the Matcher. Area is created during Initialize Matcher , passed back to the API program, and <i>must</i> be passed to each subsequent call.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

m_getmat Function

The **m_getmat** function obtains the specified record that matched successfully on a reference match from the specified window.

Syntax

```
m_getmat(RetCode, Level, Which_Match, Curr_Matrec, UserArea);
```

Example

```
m_getmat(&ret,
         &level,
         &which_match,
         curr_matrec,
         userarea);
```

Table 4.15, “Obtain Reference Match (m_getmat) Parameters (C on UNIX)” describes the parameters used by the **m_getmat** function.

Table 4.15 Obtain Reference Match (m_getmat) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code. See the “Error Codes from Obtain Reference Match” section in Chapter 4 of Volume II.
Level	Char	In	1	Indicator for match level; 0=Household or 1=Individual . <i>Commercial</i> match is the same as a <i>Household</i> match, and uses a 0 value.
Which_Match	Int	In	n*	Numerical indicator of that matched record to retrieve.

**Table 4.15 Obtain Reference Match (m_getmat)
Parameters (C on UNIX)**

Parameter	Type	Use	Length	Description
Curr_Matrec	Char	Out	n*	Buffer to receive the record from the window. Length <i>must</i> be the record length plus three bytes to hold pattern ID.
UserArea	Void **	In	16	Address of parameter area in the API; maintained by the Matcher. Created during Initialize Matcher , passed back to the program. <i>Must</i> be passed to each subsequent call.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

m_getsus Function

The **m_getsus** function obtains the specified record that suspected successfully on a reference match from the specified window.

Syntax

```
m_getsus(&ret, &level, &which_match, curr_matrec, userarea);
```

Example

```
m_getsus(&ret,
         &level,
         &which_match,
         curr_matrec,
         userarea);
```

Table 4.16 Obtain Reference Suspect Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned by the subroutine. See "Error Codes from Obtain Reference Suspect" in Chapter 4 of Volume II.
Level	Char	In	1	Match level: 0 =Household; 1 =Individual A <i>Commercial</i> match is the same as a <i>Household</i> match, and uses a 0 value.
Which_Match	Char	In	n*	Numeric; suspect matched record to retrieve.
Curr_Matrec	Char	Out	n*	Buffer to receive the record from the window. Buffer length <i>must</i> be the record length plus three bytes (lrecl +3) to hold the matched pattern ID.
UserArea	Void **	In	16	Address of parameter area that resides in the calling program and maintained by the Matcher. Created during Initialize Matcher , passed to the program. <i>Must</i> be passed to each subsequent call.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

m_params Function

The **m_params** function processes a Matcher parameter file (either a field list or a pattern list).

Syntax

m_params(RetCode, Hhld_Fname, ParmType, NULL, NULL, UserArea);

Example

```
m_params (&ret,
         hhld_fname,
         parmtime,
         NULL,
         NULL,
         userarea);
```

Table 4.15, “Obtain Reference Match (m_getmat) Parameters (C on UNIX)” describes parameters used by C calls to the **m_params** function.

**Table 4.17 Process Parameters
(m_params) Parameters (C on UNIX)**

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned. See “Error Codes from Process Parameters” in Chapter 4 of Volume II.
Hhld_Fname	Char	In	n*	Name of the Matcher parameter file to process.
ParmType	Char	In	1	Numerical indicator of the matcher parameter file to process. See the next table for the list of parameter types and their numerical identifiers.
NULL	–	–	–	Not used; pass NULLs.
NULL	–	–	–	Not used; pass NULLs.
UserArea	Void **	In	16	Address of parameter area that resides in the API program and maintained by the Matcher. Area is created during Initialize Matcher , passed back to the API program, and <i>must</i> be passed to each subsequent call.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

This table describes the parameter file types for the **m_params** function.

Table 4.18 Parameter File Types for Process_Parameters

Value	Description
0	Field list parameters for producing <i>common data segment</i> . When the 'parmtime' value is "0", the following field descriptions, along with dictionary field names or field positions and length values, should appear in the specified parameter file to allow for the creation of the common data segment. <ul style="list-style-type: none">• branch_number• middle_name• business_name• source_id• street_title• last_name• first_name• postal_code• gender• house_number
1	Household field comparison routine list.
2	Individual field comparison routine list.
3	Business field comparison routine list.
4	Household grade pattern list.
5	Individual grade pattern list.
6	Business grade pattern list.

m_retrvr Function

The **m_retrvr** function retrieves the specified relative record from a specified window record list.

Syntax

m_retrvr(WinIndicator, RecNumber, Buffer, RetCode)

Example

```
m_retrvr(&ret,
         WinIndicator,
         RecNumber,
         Buffer);
```

Table 4.19, "Retrieve Records (m_retrvr) Parameters (C on UNIX)" describes the parameters used by C calls to the **m_retrvr** function.

Table 4.19 Retrieve Records (m_retrvr) Parameters (C on UNIX)

Parameter	Type	Use	Length	Description
WinIndicator	String	In	1	Indicates which window to use; either R=Retail or C=Commercial .
RecNumber	Int	In	n ¹	Indicates the record to retrieve.
Buffer	Char	Out	n ²	Area which receives the retrieved record. If window matching (matchWindow) is used to match records, the buffer <i>must</i> be defined as an array (input record length + 184) bytes long, because the matching results are appended to the end of the record.
RetCode	Char	Out	1	Program status code.

n¹ = Length of numeric value indicating which match to retrieve.

n² = Length of the record as defined during the initializeMatcher call + 184 bytes to contain the window match key results.

m_sortwn

The **m_sortwn** function sorts the matched window records into a specified order (such as matched individuals in matched household).

Syntax

```
m_sortwn(&ret, MA_Retail_Flag, MA_Match_Sort_Key, userarea);
```

Example

```
m_sortwn(&ret,  
         MA_Retail_Flag,  
         MA_Match_Sort_Key,  
         userarea);
```

Table 4.20, "Sort Window (m_sortwn) Parameters (C on UNIX)" describes the parameters used by C calls to the **m_sortwn** function.

**Table 4.20 Sort Window
(m_sortwn) Parameters (C on UNIX)**

Parameter	Type	Use	Length	Description
RetCode	Char	Out	1	Error code returned. See the "Error Codes from Sort Window" section in Chapter 4 of Volume II.
MA_Retail_Flag	Char	In	1	Indicates the window to sort: R=Retail or C=Commercial .

**Table 4.20 Sort Window
(m_sortwn) Parameters (C on UNIX) (Continued)**

Parameter	Type	Use	Length	Description
MA_Match_Sort_Key	Char	In	1	Numerical indicator of which keys from the match to use for sort: 1 =Matched individuals within matched households 2 =Suspect individuals within suspect households 3 =Matched individuals within suspect individuals, within matched households
UserArea	Void **	In	16	Address of the parameter area residing in the API program and maintained by the matcher. Area is created during Initialize Matcher , passed back to the API program, and passed to each subsequent call.

Calling the RDI Module from C/C++

Residential Delivery Indicator (RDI) is a USPS supplied product that tells whether a record, that has been zip+4 by US Postal Matcher, it a residential delivery point. This bulletin describes how the RDI API call opens and reads US postal records and identifies whether the records are residential establishments.

The return value will be Y if the address is residential and blank (fail level 0) if it is a commercial establishment.

tsRDI

The tsRDI function is used to call the Residential Delivery Indicator module.

Syntax

```
tsRDI(
    char *calltype,
    char *recinp,
    char *ret_code,
    char *filezip9,
    char *filezip11,
```

```
char *loadmem_flag,
int zip_offset,
int ans_offset)
```

Table 4.21, “tsRDI Parameters (C++ on Windows)” describes the parameters used by C++ calls to the **tsRDI** function.

Table 4.21 tsRDI Parameters (C++ on Windows)

Parameter	Type	Description
calltype	Char	Defines the call type. Possible values: O =Open P =Process C =Close
recinp	Char	Pointer to character array that contains input and output data.
ret_code	Char	Defines the return code. Return values are defined in rdicien.h. Possible values: 0 =Normal return (no problems) 2 =Invalid calltype 3 =Error processing the open call, 4 =Memory allocation error 5 =Invalid input zip offset defined, 6 =Invalid answer offset defined, 7 =Table read error 8 = Tried to perform multi open calls without a close.
filezip9	Char	File name of the Zip9 Lookup file (rts.hs9)
filezip11	Char	File name of the Zip11 Lookup file (rts.hs11)
loadmem_flag	Char	tables into memory. Possible values: Y = Load filezip9 and filezip11 into memory (default). N = Do not load files into memory. Will read files as needed.
zip_offset		Location within recinp of the input zip11. Value is 0 based.

Table 4.21 tsRDI Parameters (C++ on Windows)

Parameter	Type	Description
ans_offset		Location within recip where the answer flag should be placed. Value is 0 based. The answer flag is one character wide.

Calling the XML APIs from C

The Trillium Software System® does not handle XML data directly; therefore, it provides a set of XML APIs for processing XML data.

► **To call the XML APIs from C**

1. Convert XML records to fixed-length records so that they can be processed by the Trillium Software System components.
2. Process the records through the Converter, Parsers, Geocoder, and Matcher.
3. Convert the final results back into XML format.

The following pseudo code shows a possible sequence of calls to process data:

```
Call xmlinit() to initialize for XML processing.  
Loop until no more XML data.  
Get an XML record.  
Call xml2fix() to convert the XML record to a fixed-length record.  
Call the Parser to parse the fixed-length record.  
Call the Geocoder to geocode the record.  
Call fix2xml() to convert the fixed-length record to XML format.  
Write the XML record to file.  
End loop.  
Call xmlend() to clean up.
```

XML Converter

The **XML Converter** program uses the following functions:

closeXMLConverter	Terminates the program and writes statistics.
getRMIErrror	Function can be called in the event of a program error so that the error message can be printed for further troubleshooting.
openXMLConverter	Reads in the parm file and keeps the parm entries in memory. Binary trees are used to hold the mappings between XML elements and DDL elements.
Fix2XML	Converts a fixed-length record to an XML-formatted record.
XML2Fix	Converts an XML formatted record to a fixed-length record.

① *Only the **openXMLConverter** function is shown.*

openXMLConverter Function

```
public int openXMLConverter(java.lang.String
parmFile,
                           java.lang.String
```

Each function uses only the parameters it needs. Table 4.22, “XML Converter Parameters” describes each parameter used in each function.

Table 4.22 XML Converter Parameters

Parameter	Type	Use	Length	Description
handle	Int	In		XMLConverter handle created during the openXMLConverter function
parmFile	String	In		Name of the XML Converter parameter file
echoFile	String	Out		Name of the file used in debugging problems with the parmFile .
XMLRec	byte []	In		Holds the input Record in an XML format.

Table 4.22 XML Converter Parameters (Continued)

Parameter	Type	Use	Length	Description
FixRec	byte []	In		Holds the output Record in fixed (text) format. Creates a buffer that holds all the XML fields with start and end tags as listed in the FIX_TO_XML_TAGS parameter in the parameter file.
retCode	byte []	Out	1	Return code returned by the program. See "XML Converter Error Codes" in Chapter 4 of Volume II.

Handle Return Values

(int) XMLConverter handle

- 1 Error; could not open the XML Converter.
- >= 0 Success; XML Converter opened with no problems.

There is a maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that *must* be preserved for each individual open. The handle contains the information required by subsequent process or close calls associated with each specific open.

Other XML Functions

The following XML functions can be called by C programs:

- xmlinit** Reads in the parameter file and stores the parameter entries in memory.
- xml2fix** Converts an XML-formatted record to a fixed-length record.
- fix2xml** Converts a fixed-length record to an XML-formatted record.
- xmlend** Performs clean up after data transformation from XML data to fixed-length data and after fixed-length data is transformed to XML.

xmlinit Function

The **xmlinit** function reads in the parameter file and stores the parameter entries in memory. Call this function once before performing any conversion.

Syntax

```
int xmlinit (char * parmfile, char * parmecho, void ** control);
```

This table describes these function parameters.

Table 4.23 Xmlinit Function Parameters

Parameter	Type	Use	Length	Description
Parmfile	char	In	n*	Name of the parameter file describing the tags for the XML-to-fixed or fixed-to-XML data conversion.
Parmecho	char	In	n*	Name of a file that is that contains the error messages regarding the parameter file. Set the value to NULL if no error messages are desired.
Control	void **	Out	16	Pointer to the control block which contains the parameter entries and other information.
retCode		Out	1	The code for the execution status.

xml2fix Function

The **xml2fix** function converts an XML-formatted record to a fixed-length record.

Syntax

```
int xml2fix (void** control, const char* xmlrec, char* fixrec);
```

Example

```
void*   control;
char    xmlrec[1024];
char    fixrec[1000];
int     retcode;
retcode = xml2fix (&control, xmlrec, fixrec);
```

This table describes these parameters.

Table 4.24 Xml2fix Function Parameters

Parameter	Type	Use	Length	Description
control	void **	In	16	Pointer to the control block which contains the parameter entries and other information.
xmlrec	char *	In	n*	XML record to be converted.
fixrec	char *	Out	n*	Converted fixed-length record.
retCode		Out	1	The execution status of the program.

fix2xml Function

The **fix2xml** function converts a fixed-length record to an XML-formatted record.

Syntax

```
int fix2xml (void** control, const char* fixrec, char* xmlrec);
```

Example

```
void*   control;
char    fixrec[1000];
char    xmlrec[1000];
int     retcode;
retcode = fix2xml (&control, fixrec, xmlrec);
```

Table 4.25, “Fix2xml Function Parameters” describes each function parameter.

Table 4.25 Fix2xml Function Parameters

Parameter	Type	Use	Length	Description
control	void**	In	16	Pointer to the control block which contains the parameter entries and other information
fixrec	char *	In	n*	Fixed-length record to convert
xmlrec	char *	Out	n*	Pointer to the XML record
retCode		Out	1	Execution status of the program

xmlend Function

The **xmlend** function performs clean up after data transformation from XML data to fixed-length data and after fixed-length data is transformed to XML.

Syntax

```
int xmlend (void** control);
```

Example

```
void*   control;  
int     retcode;  
retcode = xmlend (&control);
```

Table 4.26, “Xmlend Parameters” describes these function parameters.

Table 4.26 Xmlend Parameters

Parameter	Type	Use	Length	Description
control	Void **	In	16	Pointer to the control block which contains the parameter entries and other information.
retCode		In	1	The execution status of the program.

API Calls from C++ on UNIX

The structure used by C++ to call a Trillium Software System module is similar to the structure used the C language; the main difference is that the include statement *must* contain the extern "C" prefix to the entry point name, as shown in the following sample Converter open call.

```
extern "C"
{
    #include <paclinet.h>
    #include <cfclient.h>
    #include <pmclient.h>
    #include <maclient.h>
}
```

API Calls from Java on UNIX

This section lists examples of Java open call statements required for all Trillium Software System modules, including all postal and census Geocoders. The Java classes required to call functions are located in the base install at [./bin/pkgs/natives](#), or at [.bin/rmi](#).

For details on every call structure, see the online HTML documentation located in the base install at [./jni_api/natives/pkgs/natives](#).

Two forms of Java APIs are delivered. It is possible to use a Java program to call the Java APIs either natively, or by using Java RMI when the Trillium Software System will be installed on a machine other than the machine where the calling program resides. Separate classes for each calling method are provided in the following directories:

[tril7vnn/bin/pkgs/natives](#) Used for native calling method

[tril7vnn/bin/pkgs/rmi](#) Used for RMI calling method

nn = Trillium Software System version number.

This Java API package includes the shared objects which *must* have the location included in the user's library path.

- ① *Every open call returns to the Java program a handle that must be preserved. The handle contains the information required by subsequent process or close calls associated with each specific open. This handle is a 4-byte integer that contains pointers to the function's resources. If multiple iterations of a module are used, the handles returned during each open must be kept separate.*

For UNIX users, the **LD_LIBRARY_PATH** variable *must* be set to the directory in which the SO files are placed. The default location is **/tril7vnn/bin**, where nn is the version number. The package includes the following classes:

Table 4.27 Java Classes (Java on UNIX)

Java Class	Provides functionality for:
TrilBusinessDataParser	Business Data Parser
TrilConverter	Converter
TrilJMatcher	Matcher

Calling the Converter API from Java

The Converter open, process, and close calls all use the same calling structure and parameters. Each call uses only the parameters it requires. These three calls are:

- openConverter** Loads any table files into memory and processes the DDLs defined by the parameter file. Creates a handle, which *must* be preserved for this **open** that is returned to the calling program. The handle is then passed on subsequent **process** and **close** calls.
- process** Reads the input data, performs any specified parameter defined operations, then writes the output record to the outArea buffer.
- closeConverter** Terminates the converter program and writes the statistics file.

- ① *Only the **openConverter** call is shown. For more information about Converter functionality, see the Batch User's Guide.*

OpenConverter Sample Java Call

```
public int openConverter(java.lang.String parmFile,
                        java.lang.String echoFile,
                        byte[] inArea,
                        byte[] outArea,
                        byte[] retCode)
```

Table 4.28, “Converter Parameters (Java on UNIX)” describes the parameters used in Java calls to the Converter.

Table 4.28 Converter Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
parmFile	String	In	n*	Name of Converter parameter file.
echoFile	String	Out	n*	Name of parameter echo file used for debugging. <i>Optional.</i>
inArea	Byte	In	Variable	Buffer that contains input data to process. Length <i>must</i> equal length defined by the input DDL.
outArea	Byte	Out	Variable	Buffer that contains output data. Length <i>must</i> equal length defined by the output DDL.
retCode	Byte	Out	1	Return code indicating the status of the program call. <i>See the section "Converter Error Codes" in Chapter 4 of Volume II.</i> This code <i>must</i> contain a zero after each call.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

Handle Return Values

```
(int) converter handle
```

- 1 Cannot open Converter.
- >=0 Success.

There is a maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that must be preserved for each individual open. The handle contains the information required by subsequent process or close calls associated with each specific open.

Calling the Business Data Parser API from Java

The open, parse, and close calls all use the same calling structure and parameters. Each call uses only the parameters it requires.

- openParser** Loads Parser word pattern and city tables into memory as defined by the parameter file. Creates a handle, which *must* be preserved if multiple BDPs will be invoked, that is returned to the calling program.
- parse** Reads the name and address (INA) input data, performs any specified parameter defined operations and writes Parser output record to the **outArea** buffer.
- closeParser** Terminates Parser program and writes the statistics file.
- getErrorMsg** Function that can be called in the event of a program error so that the error message can be printed for further troubleshooting.

① *Only the **openParser** call is shown. For more information about Business Data Parser functionality, see the Batch User's Guide.*

Open Parser Sample Java Call

```
public int openParser(byte[] parmArea,
                    byte[] inArea,
                    byte[] outArea,
                    java.lang.String parmName)
```

Table 4.29 Business Data Parser Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
parmArea	Byte	Out	3,000	Contains the Business Data Parser parameter file. This area is built on the open call. The parameter file defined by the ParmName parameter is processed and then populated to parmArea . This buffer <i>must</i> be maintained after the open, then passed to each process call (parse) associated with this OpenParser instance.
inArea	Byte	In	Up to 1000	Contains Business Data Parser input data. Size is defined by the line parameters in the Business Data Parser parameter file.
outArea	Byte	Out	13,000	Business Data Parser output. <i>See the Business Data Parser PREPOS section in Chapter 4 of Volume II.</i>
parmName	String	In	n*	Name of Business Data Parser parameter file.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

Handle Return Values (int) parser handle

- **1** Error; could not open the Business Data Parser.
- >= **0** Success; Parser opened with no problems.

There is a maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that *must* be preserved for each individual open. The handle contains the information required by subsequent process or close calls associated with each specific open.

Calling the Create Window Key API from Java

This module is used to create window keys used in matching. A *window key* is a composite structure constructed from elements in the input record. The window key is then used to select records for inclusion to the Match window for comparison. This program uses a rules-based parameter file to create up to 30 window keys per record.

- open** Opens the Create Window Key program. Creates a handle, which *must* be preserved if multiple window key generators are invoked or the error message function is to be used, that is returned to the calling program.
- process** Reads the input data, performs any specified parameter defined operations, then writes the output record to the outArea buffer.
- close** Terminates the program.

❗ *Only the **open** and **process** calls are shown. For more information about Create Window Key functionality, see the Batch User's Guide.*

Create Window Key Sample Java Open Call

```
public int open(java.lang.String parmFile,  
               byte[] currentRecord,  
               byte[] windowKey)  
               int[] retCode)
```

Create Window Key Sample Java Process Call

```
public void process(int handle,  
                   java.lang.String parmFile,  
                   byte[] currentRecord,  
                   byte[] windowKey,  
                   int[] retCode)
```

Table 4.30, “Create Window Key Parameters (Java on UNIX)” describes parameters used for Java calls to Create Window Key.

Table 4.30 Create Window Key Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
handle	Int	In		Window Key handle returned by the open call.
parmFile	String	In	Variable	Array that contains the null-terminated path and file name of the parameter file.
currentRecord	Byte	In	Variable	Record containing data to derive the window key.
windowKey	Byte	Out	1500	Buffer to hold window key results, can contain 30-50 byte window keys. Keys are left-justified every 50 bytes; total of 1500 bytes.
retCode	Int	Out	1	Error code returned. See “Window Key Generator Output” in Chapter 4 of Volume II.

Handle Return Values `(int) WindowKey handle`

- 1** Error; could not open the Window Key program.
- >= 0** Success; Window Key program opened with no problems.

There is a maximum of 200 handles, each one representing a separate **open** instance. Each **open** call returns a handle that *must* be preserved for each individual **open**. The handle contains the information required by subsequent **process** or **close** calls associated with each specific **open**.

Calling the Matcher API from Java

The Matcher is different from the other module calls in that there are many calls to individual functions within the matcher module itself. The type of match you wish to perform dictates the functions you need to call. *Reference Matching* and *Window Matching* and the operation of each are detailed below. For information about Matcher functionality, see the *Batch User’s Guide*.

Reference Matching Mode

This section discusses the steps required to use the Matcher in the Reference Match mode. This mode of operation matches one record (called the *candidate record*) to one or more records (called the *reference record*) that have been retrieved from a source like a database. The match rules are contained in external parameter files that are loaded to memory upon initialization.

The following requirements must be met for a Reference match:

- Both the candidate record and reference records *must* be the same shape.
- Reference records *must* be processed through the same or similar steps as the candidate record to ensure field standardization.
- Matcher fields list parameter files *must* be configured to use field locations (**fldlocs**) and *not* DDL field names (as is commonly used in batch mode).
- If an optional candidate matching information key is used, the reference records *must* contain a unique key for both household and individual levels.

Reference Match Step and Module Names

The table below describes a set of minimum steps required to perform a Reference match and retrieve the results. For more information about each module, see the appropriate section of this document.

- ① *Steps external to the Trillium Software System, but necessary to the match process, are **not** included in this list; for example, a step which pulls records from a database to load to the Match window would be performed by either the calling program or another external subroutine.*

Table 4.31, “Suggested Reference Match Function Name Order (Java on UNIX)” describes the Reference match function names in suggested order.

Table 4.31 Suggested Reference Match Function Name Order (Java on UNIX)

Entry point	Description
initializeMatcher	Used to set up Match windows and initialize the Matcher.
processParameters	Each call <i>must</i> process one of the required fields or patterns parameter files required for the Matcher. Field files <i>must</i> use the fldlocs keyword instead of the fldnames keyword because DDLs are not used in the calling process. Call this function four times for a retail match; twice for a commercial match.
addRecord	Loads Match window with reference records (number of records with window key of the same value as the candidate record)
matchCandidate	Used to match a transaction record to the database records that have been loaded to the Match window during addRecord .
numberRefmatches	Used to obtain the number of records that matched the candidate record.
obtainRefmatch	Used to return a value in Number of Reference Matches that is equal to the total number of reference match records that are available to be obtained (one record is called at a time.) <ul style="list-style-type: none"> • Each individual record is written to a buffer. That record must be moved from the buffer before calling for another record. Otherwise, original record is overwritten. • Buffer is the input record length plus 3 bytes holding the match pattern ID number (for example, P150 would have a pattern ID of 150). • By default, the returned buffer is sorted in match pattern ID ascending order. If this order is not desired, the pattern ID numbers can all be set to the same value. • Calling parameters specify if you are requesting a household/commercial match or an individual match.
clearWindow	Used to empty the Match window, and restart, to match the next transaction.
endMatcher	After all matching is complete, this is used to close the Matcher and write statistics.

- ① *Suspect matches can be retrieved in a similar way if suspect match patterns are being used. Call **numberRefsuspects** and use the value returned to loop for calling **obtainRefsuspects**.*

Window Matching

This section discusses the steps required to use the Matcher in the *Window Match* mode. This mode of operation matches a group of records to one another. The match rules are contained in external parameter files that are loaded to memory upon initialization.

Window Match Step and Module Names

Table 4.32, "Suggested Function Name Order (Java on UNIX)" lists the minimum steps to perform a Window match.

- ① *Steps external to the Trillium Software System, but necessary to the match process, are **not** shown. A step which pulls records from a database to load to the Match window would be performed by either the calling program or by another external subroutine.*

Table 4.32 Suggested Function Name Order (Java on UNIX)

Name	Description
initializeMatcher	Used to set up the Match windows and initialize the Matcher.
processParameters	Each call <i>must</i> process one of the required fields or patterns parameter files required for the Matcher. Field files <i>must</i> use the fdlocs keyword because DDLs are not used in the calling process. Call this function four times for a retail match; twice for a commercial match.
addRecord	Used to load the Match window with reference records (the number of records with window key of the same value as the candidate record).
matchWindow	Used to match the records that have been loaded to the Match window during addrecord .
createCommon	Used to append the common data and survivor flag information on the matched records (<i>available only in window match</i>). <i>Optional</i> .

Table 4.32 Suggested Function Name Order (Java on UNIX)

Name	Description
retrieveRecords	Used to find the number of window records to retrieve the matched record with appended match information. Returned buffer <i>must</i> be input record length + 184 bytes.
sortWindow	Used to sort the matched window records into a specified order (such as matched individuals within a matched household).
clearWindow	Used to empty the Match window, and restart with Add Record .
endMatcher	After all matching has been completed, closes the Matcher and writes statistics.

The following functions are all used as part of the Matcher.

initializeMatcher Function

The **initializeMatcher** function creates the Match window areas for either a reference match or a window match.

```
public char initializeMatcher(java.lang.String dataFile,
                             java.lang.String linkFile,
                             int recLength,
                             int retCode)
```

Table 4.33, “initializeMatcher Parameters (Java on UNIX)” describes the parameters used by Java calls to the Matcher.

Table 4.33 initializeMatcher Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
DataFile	String	In	n*	Name of file to display program statistics.
LinkFile	String	In	n*	File to contain keys of matched records when performing multiple window match scenarios. Window matching only.
RecLength	Int	In	n*	Input file record length to match. A negative length activates Field matcher tracing.
RetCode	Int	Out	1	See “Error Codes from Initialize Matcher” in Chapter 4 of Volume II.

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

Handle Return Values

- 1 Error; could not open the Matcher program.
- >= 0 Success; Matcher program opened with no problems.

processParameters function

The **processParameters** function processes a Matcher parameter file (either a field list or a pattern list).

```
public char processParameters(int handle,
                             java.lang.String parmFile,
                             java.lang.String parmType)
```

Table 4.34, “processParameters Parameters (Java on UNIX)” describes parameters used by Java calls to the **processParameters** function.

Table 4.34 processParameters Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open. The handle contains information required by subsequent process or close calls associated with each specific open.
parmFile	String	In	n*	Name of the parameter file to process.
parmType	String	In	1	Parameter file type to process. <i>See the next table.</i>

n* = Platform-specific width of a file name. Note that all file names should terminate with a null character after the last valid file name character.

Table 4.35, “Possible Types of Parameter Files for Process_Parameters (Java on UNIX)” describes the parameter file types used by Java calls to the **Process_Parameters** function.

Table 4.35 Possible Types of Parameter Files for Process_Parameters (Java on UNIX)

Value	Description
0	Field list parameters for producing the Common Data Segment. (Optional for window matching.) The following field descriptions, along with dictionary field names or field positions and length values, should appear in the specified parameter file to allow for the creation of the <i>Common Data Segment</i> . <ul style="list-style-type: none"> • branch_number • middle_name • business_name • last_name • first_name • postal_code • gender • street_title • house_number • source_id
1	Household field comparison routine list. REQUIRED for retail matching
2	Individual field comparison routine list. REQUIRED for retail matching
3	Business field comparison routine list. REQUIRED for commercial matching
4	Household grade pattern list. REQUIRED for retail matching
5	Individual grade pattern list. REQUIRED for retail matching
6	Business grade pattern list. REQUIRED for commercial matching
7	Field list parameters for producing candidate matching information. (Optional for reference matching.) The following field descriptions, along with dictionary field names or field positions and length values, should appear in the specified parameter file to allow for the creation of the matched information block. <ul style="list-style-type: none"> • household_number • individual_number • record_id

addRecord Function

The **addRecord** function adds the specified record to a specified window record list.

```
public char addRecord(int handle,
                     java.lang.String winIndicator,
                     byte[] record)
```

Table 4.36, “addRecord Function Parameters (Java on UNIX)” describes the parameters used by Java calls to the Matcher.

Table 4.36 addRecord Function Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open . Handle contains information required by subsequent process or close calls associated with each specific open .
WinIndicator	String	In	1	Indicates the window to expand; either R=Retail or C=Commercial
Record	Byte	In	n*	Specifies the record to add.

n* = Length of the record to be added to the match window. The length used here must be the same as what was defined during the initializeMatcher call.

- ① See “Error Codes from Add Records” in Chapter 4 of Volume II for a list of error codes returned by this Matcher function.

matchCandidate Function

The **matchCandidate** function initiates the matching of a specified candidate record against records in the specified window record list and saves information about the best matches (reference match).

```
public char matchCandidate(int handle,
                           java.lang.String winIndicator,
                           byte[] record)
```

Table 4.37, “matchCandidate Parameters(Java on UNIX)” describes the parameters used by Java calls to matchCandidate.

Table 4.37 matchCandidate Parameters(Java on UNIX)

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open . The handle contains information required by subsequent process or close calls associated with each specific open .
winIndicator	String	In	1	The window to match, and matching type: R=Retail (use both the Retail window and the Retail field and pattern lists) C=Commercial (use both Commercial window and Commercial field and pattern lists).
record	Byte	In	n*	Specifies the candidate record to match.

n* = Length of the candidate record to be matched to the records added to the Match window during the **addRecord** call. The length used here must be the same as what was defined during the **initializeMatcher** call.

numberRefMatches Function

The **numberRefMatches** function finds the number of matches which occurred on a reference match.

```
public long numberRefMatches (int handle,
                             java.lang.String matchLevel)
```

This table describes parameters used by Java calls to the **numberRefMatches** function.

Table 4.38 Number of Reference Matches Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open. The handle contains information required by subsequent process or close calls associated with each specific open.
MatchLevel	String	In	1	Values include: 0=Household/Business or 1=Individual

- ① *See the section "Error Codes from Number of Reference Matches" in Chapter 4 of Volume II for a list of error codes returned by this Matcher function.*

numberRefSuspects Function

The **numberRefSuspects** function finds the number of suspects matches which occurred on a reference match.

```
public long numberRefSuspects(int handle,
                               java.lang.String matchLevel)
```

Table 4.39, “Number of Reference Suspects (numberRefSuspects) Parameters (Java on UNIX)” describes parameters used for Java calls to the **numberRefSuspects** function.

**Table 4.39 Number of Reference Suspects
(numberRefSuspects) Parameters (Java on UNIX)**

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open. The handle contains information required by subsequent process or close calls associated with each specific open.
MatchLevel	String	In	1	Values include: 0=Household/Business or 1=Individual

- ① *See the section “Error Codes from Number of Reference Suspects” in Chapter 4 of Volume II for a list of error codes returned by this Matcher function.*

obtainRefMatch function

The **obtainRefMatch** function obtains the specified record that matched successfully on a reference match from the specified window.

```
public char obtainRefMatch(int handle,
                           java.lang.String matchLevel,
                           int whichMatch,
                           byte[] buffer)
```

Table 4.40, “Obtain Reference Match (obtainRefMatch) Parameters (Java on UNIX)” describes the parameters used by Java calls to the **obtainRefMatch** function.

Table 4.40 Obtain Reference Match (obtainRefMatch) Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open .
matchLevel	String	In	1	Set to either: 0=Household/Business or 1=Individual
whichMatch	Int	In	n ¹	Numeric value indicating which match to retrieve from the Match window.
buffer	String	Out	n ²	Buffer to contain the record retrieved from the match window during the ObtainRefMatch call.

n¹ = Length of numeric value indicating which match to retrieve.

n² = Length of the record as defined during the **initializeMatcher** call + 3 bytes to contain the matched pattern ID.

- ① *See the section “Error Codes from Obtain Reference Match” in Chapter 4 of Volume II for a list of error codes returned by this Matcher function.*

obtainRefSuspect Function

The **obtainRefSuspect** function obtains the specified record that suspected successfully on a reference match from the specified window.

```
public char obtainRefSuspect(int handle,
                             java.lang.String matchLevel,
                             int whichSuspect,
                             byte[] buffer)
```

Table 4.41, "Obtain Reference Suspect (obtainRefSuspect) Parameters (Java on UNIX)" describes the parameters used in Java calls to the **obtainRefSuspect** function.

Table 4.41 Obtain Reference Suspect (obtainRefSuspect) Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open . The handle contains information required by subsequent process or close calls associated with each specific open .
matchLevel	Int	In	1	Values either 0=Household/Business or 1=Individual .
whichSuspect	Int	In	n ¹	A numeric value indicating which suspect match to retrieve from the suspect match window.
buffer	String	Out	n ²	Buffer containing record retrieved from suspect match window during ObtainRefMatch call.

n¹ = Length of numeric value indicating which suspect match to retrieve.

n² = Length of the record as defined during the **initializeMatcher** call + 3 bytes to contain the suspect pattern ID.

sortWindow Function

The **sortWindow** function sorts the matched window records into a specified order (for example, matched individuals in matched household).

```
public char sortWindow(int handle,
                       java.lang.String winIndicator,
                       java.lang.String sortKey)
```

Table 4.42, "Sort Window (sortWindow) Parameters (Java on UNIX)" describes the parameters used by Java calls to **sortWindow**.

**Table 4.42 Sort Window
(sortWindow) Parameters (Java on UNIX)**

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open . The handle contains information required by subsequent process or close calls associated with each specific open .
winIndicator	String	In	1	Indicates the window to use: either R=Retail , or C=Commercial .
sortKey	String	In	1	Indicates the keys to use for the sort: 1 =Matched individuals in matched households. 2 =Suspect individuals in suspect households. 3 =Matched individuals in suspect individuals within matched households.

① *See the section "Error Codes from Sort Window" in Chapter 4 of Volume II for a list of error codes returned by this Matcher function.*

matchWindow function

The **matchWindow** function initiates the matching of a specified window record list against itself. Appends the results of the matching process to each record in the record list (Window match).

```
public char matchWindow(int handle
                        java.lang.String winIndicator)
```

Table 4.43, “Match Window (matchWindow) Parameters (Java on UNIX)” describes parameters used by Java calls to **matchWindow**.

**Table 4.43 Match Window
(matchWindow) Parameters (Java on UNIX)**

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open . The handle contains information required by subsequent process or close calls associated with each specific open .
winIndicator	String	In	1	The window to match, and type of matching; either R=Retail (use both the Retail window and the Retail field and pattern lists) or C=Commercial (use both the Commercial window and the Commercial field and pattern lists).

- ① *See the section “Error Codes from Match Window” in Chapter 4 of Volume II for a list of error codes returned by this Matcher function.*

createCommon Function

The **createCommon** function creates the *common data segment* for matched records in a record list. The common data is built of the "best" data from matches based on the rules defined in the common fields parameter file. Also, produces the linked records file, if desired. (*Window Matching Only*)

```
public char createCommon(int handle,
                        java.lang.String winIndicator)
```

Table 4.44, "createCommon Parameters (Java on UNIX)" describes the parameters used by Java calls to the Matcher.

**Table 4.44 createCommon Parameters
(Java on UNIX)**

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open . The handle contains information required by subsequent process or close calls associated with each specific open .
WinIndicator	String	In	1	Indicates which window is to be cleared; either R=Retail or C=Commercial

- ① *See the section "Error Codes from Create Common" in Chapter 4 of Volume II for a list of error codes returned by this Matcher function.*

retrieveRecords Function

The **retrieveRecords** function retrieves the specified relative record from a specified window record list.

```
public char retrieveRecords(int handle,
                           java.lang.String winIndicator,
                           int recNumber,
                           byte[] buffer)
```

Table 4.45, "Retrieve Records (retrieveRecords) Parameters (Java on UNIX)" describes parameters used by Java calls to the **retrieveRecords** function.

**Table 4.45 Retrieve Records
(retrieveRecords) Parameters (Java on UNIX)**

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open . The handle contains information required by subsequent process or close calls associated with each specific open .
winIndicator	String	In	1	Indicates window: either R=Retail , or C=Commercial
recNumber	Int	In	n ¹	Indicates the record to retrieve.
buffer	byte	Out	n ²	Area that receives the retrieved record.

n¹ = Length of numeric value indicating which match to retrieve.

n² = Length of the record, as defined during the **initializeMatcher** call + 184 bytes to contain the window match key results.

- ① See the section "Error Codes from Retrieve Records" in Chapter 4 of Volume II for a list of error codes returned by this Matcher function.

clearWindow Function

The **clearWindow** function erases the specified window record list and can optionally produce window statistics.

```
public char clearWindow(int handle,
                        java.lang.String winIndicator)
```

Table 4.46, “Clear Window Parameters (Java on UNIX)” describes parameters used by Java calls to the ClearWindow function.

Table 4.46 Clear Window Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open . The handle contains information required by subsequent process or close calls associated with each specific open .
WinIndicator	String	In	1	Indicates which window is to be expanded; either R=Retail or C=Commercial .

- ① *See the section “Error Codes from Clear Window” in Chapter 4 of Volume II for a list of error codes returned by this Matcher function.*

endMatcher Function

The **endMatcher** function optionally displays the program statistics, erases both the retail and commercial window record lists if they exist, and terminates the Matcher.

```
public char endMatcher(int handle)
```

Table 4.47, “End Matcher Parameters (Java on UNIX)” describes the parameters used by Java calls to the Matcher.

Table 4.47 End Matcher Parameters (Java on UNIX)

Parameter	Type	Use	Length	Description
handle	Int	In	--	Maximum of 200 handles, each one representing a separate open instance. Each open call returns a handle that <i>must</i> be preserved for each individual open . The handle contains information required by subsequent process or close calls associated with each specific open .

- ① *See the section “Error Codes from End Matcher” in Chapter 4 of Volume II for a list of error codes returned by this Matcher function.*

Index

A

- ACTION parameter 3-3, 3-29
- Add Record function (C on UNIX) 4-9, 4-11
- addRecord function (Java on UNIX) 4-47, 4-48
- addRecord function parameters (Java on UNIX) 4-52
- API calls (C++ on UNIX) 4-39

B

- Business Data Parser
 - calling from C 4-5
 - Calling from C# 3-5
 - calling from JAVA 4-42
 - Calling from VB 3-55
 - Calling from VB. NET 3-31

C

- C calls on Windows 4-2
- C/C++ Programs
 - API calls from 3-2
- calling the Customer Data Parser from C# 3-7
- calling the Trillium Software System modules from C or C++ 3-2, 4-2
- CandidateCode1Routine parameter 2-8
- cf_coni function parameters (C on UNIX) 4-4
- cf_coni function syntax (C on UNIX) 4-4
- cf_crwkey function
 - Declare statement for (VB) 3-56, 3-57
 - parameters (C#) 3-6

- cf_crwkey function (C on UNIX) 4-7
- cf_crwkey function parameters (VB) 3-57
- cf_crwkey function syntax (C on UNIX) 4-7
- cfclient.h header file 4-2
- cflib.dll 3-3
- Clear Window function (C on UNIX) 4-10, 4-11
- clearWindow function (Java on UNIX) 4-47, 4-49
- closeCGeocoder function (Java on UNIX) 4-33
- closeParser call (Java on UNIX) 4-42
- Converter
 - Calling from C# 3-4
 - calling from JAVA 4-40
 - Calling from VB 3-53
 - Calling from VB. NET 3-30
 - closeConverter call (Java) 4-40
 - openConverter call (Java) 4-40
 - process call (Java) 4-40
- Converter parameters (Java on UNIX) 4-41
- Country Data Router
 - Sample Open Call 3-52
- Create Common function (C on UNIX) 4-11
- Create Window Key
 - Calliing from C 4-7
 - calling from JAVA 4-44
 - Calling from VB 3-56
 - Calling from VB. NET 3-32
 - function prototype for calling (C#) 3-6
 - sample Java Open call (JAVA) 4-44
- Create Window Key Function
 - parameter files for (C on UNIX) 4-7
- Create Window Key parameters (Java

- on UNIX) 4-45
- createCommon function (Java on UNIX) 4-48
- creating prototypes
 - for the unmanaged functions 3-2
- Customer address
 - specifying 2-7
- Customer Data Parser
 - Calling from C# 3-7
 - calling from C# 3-7
- Customer name
 - specifying 2-7
- D**
- Data Reconstructor
 - Calling from VB. NET 3-33
- Data Reconstructor parameter file
 - specifying name and location of 2-8
- data transformation
 - cleaning up from 4-35
- debugging
 - using Data Reconstructor echo file for 2-8
 - using Router echo file for 2-8
 - using USDPV echo file for 2-10
- Delivery address format
 - specifying 2-9
- DllImport attribute 3-2
- DLLs
 - required for calling Trillium Software System modules 3-3
- DPV parameters
 - See USDelivery Point Validation parameter file 2-10
- dynamic linking
 - to Trillium Software System libraries 4-2

- E**
- End Matcher function (C on UNIX) 4-10, 4-11
- endMatcher function (Java on UNIX) 4-47, 4-49
- extern "C" prefix
 - using, in C++ calls 4-39
- extern modifier
 - for DllImport attribute 3-2
- F**
- fix2xml function syntax (C on UNIX) 4-37
- fixed-length records
 - converting to XML-formatted records 4-35
- Form file name 2-7
- G**
- ga_prsi function parameters (C on UNIX) 4-6
- gaclient.h header file 4-2
- galib.dll 3-3
- getErrorMsg call (Java on UNIX) 4-42
- getErrorMsg function (Java on UNIX) 4-33
- H**
- header files
 - list of 4-2
- I**
- Initialize Matcher function (C on UNIX) 4-9, 4-11
- initializeMatcher function (Java on UNIX) 4-47, 4-48
- INP_DDL parameter file (C on UNIX) 4-7

J

Java API package 4-40
Java Classes (Java on UNIX) 4-40
Java open call statements
 examples of, locations 4-39

L

libraries
 for C calls, location of 4-2
linking to the Trillium Software
 System libraries
 dynamically 4-2
 statically 4-2
List file
 specifying name of 2-9

M

m_addrac function (C on UNIX) 4-9,
 4-11, 4-12
m_addrac function (VB) 3-59, 3-60
m_clearw function (C on UNIX) 4-10,
 4-11
m_clearw function (VB) 3-59, 3-61
m_cmatch function (C on UNIX) 4-9
m_cmatch function (VB) 3-59
m_common function (C on UNIX) 4-11
m_common function (VB) 3-60
m_endmat function (C on UNIX) 4-10
m_endmat function (VB) 3-59, 3-61
m_getmat function (C on UNIX) 4-10
m_getmat function (VB) 3-59
m_getmat function parameters (C on
 UNIX) 4-22
m_getsus function parameters (C on
 UNIX) 4-24
m_initmr function
 parameters (C#) 3-12
m_initmr function (C on UNIX) 4-9,
 4-11

m_initmr function (C#) 3-11
m_initmr function (VB) 3-58, 3-60
m_nummat function parameters (C on
 UNIX) 4-20
m_nummat function (C on UNIX)
 4-10, 4-19
m_nummat function (VB) 3-59
m_numsus function (C on UNIX) 4-21
m_numsus function parameters (C on
 UNIX) 4-21
m_params function (C on UNIX) 4-9
m_params function (VB) 3-59, 3-60
m_params function parameters (C on
 UNIX) 4-25
m_paramsfunction (C on UNIX) 4-11
m_retrvr function (C on UNIX) 4-11,
 4-27
m_retrvr function (VB) 3-60
m_retrvr function parameters (C on
 UNIX) 4-27
m_sortwn function (C on UNIX) 4-11
m_sortwn function parameters (C on
 UNIX) 4-28
m_wmatch function (C on UNIX) 4-11,
 4-18
m_wmatch function (VB) 3-60
m_wmatch function parameters (C on
 UNIX) 4-19
maclient.h header file 4-2
managed code 3-2
Match Candidate function
 parameters (C#) 3-16
Match Candidate function (C on UNIX)
 4-9
Match Window function (C on UNIX)
 4-11
matchCandidate function (Java on
 UNIX) 4-47
Matcher

- Add Record (addRecord) function (JAVA) 4-52
- Add Record (m_addrec) function (VB. NET) 3-40
- Add Record (m_addrec) function (VB) 3-62
- Calling from C 4-8
- Calling from C# 3-7
- calling from JAVA 4-45
- Calling from VB 3-57
- Calling from VB. NET 3-33
- Clear Window (addRecord) function (JAVA) 4-62
- Clear Window (m_clearw) (C) 4-13
- Clear Window (m_clearw) function (C#) 3-15
- Clear Window (m_clearw) function (VB. NET) 3-41
- Clear Window (m_clearw) function (VB) 3-63
- Create Common (m_common) function 3-23
- Create Common (m_common) function (VB. NET) 3-48
- Create Common (m_common) function (VB) 3-64
- Create Common Data Segment (m_common) (C) 4-14
- Create Common function (JAVA) 4-60
- End Matcher (m_endmat) function 3-21
- End Matcher (m_endmat) function (VB. NET) 3-47
- End Matcher (m_endmat) function (VB) 3-65
- End Matcher function (JAVA) 4-63
- Initialize Match Window (m_initmr) (C) 4-16
- Initialize Matcher (m_initmr) function (C#) 3-11
- Initialize Matcher (m_initmr) function (VB. NET) 3-37
- Initialize Matcher (m_initmr) function (VB) 3-66
- Initialize Matcher function (JAVA) 4-49
- Match Candidate (m_cmatch) function (C#) 3-16
- Match Candidate (m_cmatch) function (VB) 3-67
- Match Candidate function (JAVA) 4-53
- Match Window (m_cmatch) function (VB. NET) 3-42
- Match Window (m_wmatch) function (VB. NET) 3-49
- Match Window (m_wmatch) function (VB) 3-68
- Match Window function (JAVA) 4-59
- Number of Reference Matches (m_nummat) function (C#) 3-17
- Number of Reference Matches (m_nummat) function (VB. NET) 3-43
- Number of Reference Matches (m_nummat) function (VB) 3-69
- Number of Reference Matches function (JAVA) 4-54
- Number of Reference Suspects (m_numsus) function (C#) 3-18
- Number of Reference Suspects (m_numsus) function (VB) 3-70

- Number of Reference Suspects
 - function (JAVA) 4-55
 - Number of Suspect Matches
 - (m_numsus) function (VB. NET) 3-44
 - Obtain Reference Match
 - (m_getmat) function (C#) 3-19
 - Obtain Reference Match function (JAVA) 4-56
 - Obtain Reference Matches
 - (m_getmat) function (VB. NET) 3-45
 - Obtain Reference Matches
 - (m_getmat) function (VB) 3-71
 - Obtain Reference Suspect
 - (m_getsus) function 3-20
 - Obtain Reference Suspect function (JAVA) 4-57
 - Obtain Reference Suspect Matches (m_getsus) function (VB. NET) 3-46
 - Obtain Reference Suspects
 - (m_getsus) function (VB) 3-72
 - Process Parameters (m_params) function (C#) 3-13
 - Process Parameters (m_params) function (VB. NET) 3-38
 - Process Parameters (m_params) function (VB) 3-73
 - Process Parameters function (JAVA) 4-50
 - Retrieve Record (m_retrvr) function 3-26
 - Retrieve Record (m_retrvr) function (VB. NET) 3-50
 - Retrieve Record (m_retrvr) function (VB) 3-75
 - Retrieve Records function (JAVA) 4-61
 - Sort Window (m_sortwn) function 3-25
 - Sort Window (m_sortwn) function (VB. NET) 3-51
 - Sort Window (m_sortwn) function (VB) 3-76
 - Sort Window function (JAVA) 4-58
 - Matcher Field List parameter files 3-58
 - matching
 - using level match values for 2-8
 - matchWindow function (Java on UNIX) 4-48
 - mllib.dll 3-3
 - Module names
 - Reference matching mode (C on UNIX) 4-9
 - Window matching mode (C on UNIX) 4-11
- N**
- native calling method (Java on UNIX) 4-39
 - Number of Reference Matches
 - function parameters (C#) 3-17
 - Number of Reference Matches
 - function (C on UNIX) 4-10
 - Number of Reference Suspects
 - function parameters (C#) 3-18
- O**
- Obtain Reference Match function (C on UNIX) 3-9, 3-35, 4-10, 4-47
 - obtainRefmatch function (Java on UNIX) 4-47

openCGeocoder function (Java on UNIX) 4-33
openParser call (Java on UNIX) 4-42

P

Parameter Initialization File
Parameters (table) 2-7
parmfile.ini file 2-7
parmfile.ini parameters
country-specific 2-9
parse call (Java on UNIX) 4-42
Parser parameter file 2-7
parsing
and Candidate Code routines 2-8
postal directory
matching street information to 2-9
matching street names to 2-10
Postal Geocoder tables
specifying location of 2-7
Process Parameters function (C on UNIX) 4-9, 4-11
Process_Parameters 4-24
Processor name
specifying 2-7
processParameters function (Java on UNIX) 4-47, 4-48

R

ReconsEchoFile parameter 2-8
ReconsParmFile parameter 2-8
Reference match function
suggested name order for (Java on UNIX) 4-47
Reference match function name
suggested order of (C on UNIX) 4-9
Reference Match requirements 3-8
Reference Match Step Names 3-58
Reference Match Step Names (VB)

3-58
Reference matching
requirements for (C on UNIX) 4-8
Reference Matching (C#) 3-7
Reference Matching (VB) 3-57
Reference Matching mode
steps for using 3-7
Reference matching mode (C on UNIX) 4-8
Reference matching mode (JAVA) 4-46
Retrieve Relative Record function (C on UNIX) 4-11
retrieveRecords function (Java on UNIX) 4-49
RMI calling method (Java on UNIX) 4-39
Router parameter file
specifying name and location of 2-8
RouterEchoFile parameter 2-8
RouterParmFile parameter 2-8
Rules file
specifying name of 2-9
RULES_DDNAME parameter file (C on UNIX) 4-7
rules-based parameter file
using to create window keys 3-6

S

sample programs
for C/C++
location in Trillium Software System installation 3-2, 4-2
Sort Window function (C on UNIX) 4-11
sortWindow function (Java on UNIX) 4-49

spelling algorithm 2-10
 used by US Centroid Census
 Geocoder 2-10

static linking
 to Trillium Software System
 libraries 4-2

static modifier
 for DllImport attribute 3-2

Step names
 Reference matching mode (C on
 UNIX) 4-9

step names
 for Reference matching mode
 (Java on UNIX) 4-46
 for Window matching mode (Java
 on UNIX) 4-48

street information
 matching to postal directory 2-9

street names
 matching to postal directory 2-10

Suspect matches
 retrieving (C on UNIX) 4-10

T

TrilBusinessDataParser class (Java on
 UNIX) 4-40

TrilConverter class (Java on UNIX)
 4-40

TrilJMatcher class (Java on UNIX) 4-40

Trillium Software System installation
 .€ib or .cdll directories in 4-2

U

UK Geocoder
 Calling from a Java Program 4-45

US Census Geocoder tables
 specifying location of 2-9

US Delivery Point Validation echo file
 2-10

US Delivery Point Validation
 parameter file
 specifying name and directory
 location of 2-10

US Interpolated Rooftop Census
 Geocoder
 specifying directory location of 2-9

USCensusDisplay BaseData
 parameter 2-10

USCensusPath parameter 2-9

USDPEchoFile parameter 2-10

USHouseHoldDisplay BaseData
 parameter 2-10

USHouseHoldPath parameter 2-9

W

window key
 defined (C#) 3-6

Window Match function names
 suggested order of (C on UNIX)
 4-11

Window match function names
 suggested order for (Java on
 UNIX) 4-48

Window Match function names (VB)
 3-60

Window Match mode 3-60

Window Match Step Names and
 Module Names 3-60

Window Matching 3-9, 3-60

Window Matching (C#) 3-7

Window matching mode (Java on
 UNIX) 4-48

WinkeyParmFile parameter 2-9

X

XML APIs
 calling from C programs 4-32
XML data

- indirectly handling 4-32
- processing 4-32
- XML Utilities for C
 - fix2xml 4-33, 4-37
 - xml2fix 4-36
 - xmlend 4-38
 - xmlinit 4-35
- xml2fix function syntax (C on UNIX)
4-36
- xmlend function syntax (C on UNIX)
4-38
- XML-formatted records
 - converting to fixed-length 4-35
- xmlinit function syntax (C on UNIX)
4-35
- XXCustomerAddress(1-4) parameter
2-7
- XXCustomerName parameter 2-7
- XXDeliveryType parameter 2-9
- XXFormFileName parameter 2-7
- XXLevelMatch parameter 2-8
- XXListName parameter 2-9
- XXMaxNames parameter 2-8
- XXParserParmFile parameter 2-7
- XXPostalDisplay BaseData parameter
2-8
- XXPostalPath parameter 2-7
- XXProcessorName parameter 2-7
- XXRulesFileName parameter 2-9
- XXService BureauName parameter
2-7