COGNOS®

Cognos Impromptu (R)

How to Create User-Defined Functions

Product Information

This document applies to Impromptu (R) Version 7.1 and may also apply to subsequent releases. To check for newer versions of this document, visit the Cognos support Web site (http://support.cognos.com).

Copyright

Table of Contents

Welcome

What Is In This Document

This document provides you with the detailed information you need to create User-Defined Functions (UDFs) for use with your Cognos product:

- Chapter 1, "UDFs at a Glance," offers general information about creating UDFs and the supported operating systems. This chapter also contains a quick reference for the steps you will follow to create UDFs.
- Chapter 2, "Modify the .ini Files," details the changes you need to make to the .ini files.
- Chapter 3, "Implement Database UDFs," discusses how to set up UDFs for your database system or systems.
- Chapter 4, "Implement External UDFs," discusses how to set up a UDF in a DLL (for Windows environments) or a shared library (for UNIX environments).
- The Appendix contains supplementary information about filename conventions, the files in the UDF Software Developer's Kit (SDK), and the cogudf.exe utility for checking the syntax of your .sql files.

What You Need to Know to Use This Document Effectively

To create User-Defined Functions (UDFs), you need to be familiar with

- using a text editor to modify text files (.ini and .sql)
- adding functions to your database (for database UDFs)
- writing, compiling and linking programs in the C language (for external UDFs)

Software Developer's Kit

You will need access to the Software Developer's Kit (SDK) for UDFs. The SDK enables you to create database UDFs and external UDFs for Windows and UNIX.

By default, the SDK is installed in the *installation_location*\cer*n*\bin\User Defined Functions folder.

The SDK contains the following files:

- this document
- the C source files necessary for implementing external UDFs
- a utility to help you quickly check the UDF definition syntax in .sql files
- sample files and Readme text files that describe how to use the sample files

For information about the files in the SDK, see the Appendix on (p. 35).

The SDK files and samples were created with Microsoft Developer Studio 97 with Visual C++ 4.2.

Conventions Used in This Document

- Code samples appear in the `Courier` font.
- To avoid confusion between Windows environments and UNIX environments, all filenames in this document are shown in full lowercase unless case-sensitivity requires otherwise.

Other Information

Our documentation includes user guides, tutorial guides, reference books, and other pieces to meet the needs of our varied audience.

All information is available in online help. Online help is available from the Help button in Windows products.

The information in each online help system is available in online book format (PDF). However, the information from a given help system may be divided into more than one online book. Use online books when you want a printed version of a document or when you want to search the whole document. You can print selected pages, a section, or the whole book. Cognos grants you a non-exclusive, non-transferable license to use, copy, and reproduce the copyright materials, in printed or electronic format, solely for the purpose of providing internal training on, operating, and maintaining the Cognos software.

In Windows products, online books are available from the Windows Start menu (Cognos) and from the product Help menu (Books for Printing). All online books are available on the Cognos documentation CD. You can also read the product readme files and the installation guides directly from the Cognos product CDs.

Only the installation guides are available as printed documents.

An annotated list of other documentation, the *Documentation Roadmap*, is available from the Windows Start menu or the Impromptu Help menu.

Questions or Comments?

For the fastest response to questions about using Impromptu, contact customer support.

For information about customer support locations and programs, see Cognos on the Web on the Help menu or visit the Cognos Support Web site (http://support.cognos.com).

Chapter 1: UDFs at a Glance

In this chapter you will find

- general notes about UDFs
- a description of the two types of UDFs and how you can use them in your reporting environment
- information about the supported operating systems
- the steps to create UDFs

Overview

You can create UDFs to

- provide a custom or business function for your reporting environment that is not already packaged with your Cognos product
- implement a function in a database as a central repository
- access bit masks or other applications that encode and package data
- work with user-defined data types found in databases such as Informix, DB2, and Oracle.

Types of UDFs

There are two types of UDFs:

- database functions

  Database UDFs are functions that you can define for use with one or more database systems. Refer to your RDBMS documentation for information about using UDFs in your database environment.

  Note: In Cognos Architect, database functions are referred to as stored procedures.
- external functions

  External UDFs are written in C or another language (the C calling convention must be supported) and are compiled into DLLs for use with Windows NT. For use in UNIX environments, external UDFs are compiled into shared libraries.

Cognos products support scalar UDFs — those that return a single value each time they are invoked.

How Cognos Products Resolve Function Names

To resolve a function name in an SQL statement, Cognos products search in the following order:

1. database functions (system and UDF) specific to a particular instance of a type of database
2. database functions specific only to the type of database
3. external UDFs
4. functions packaged with your Cognos product

Important!

- You should back up all the UDF .ini and .sql files you modify when working with the samples or creating your own UDFs. These files will be overwritten if you re-install your Cognos product, and any changes you may want to keep will be lost.
- To easily migrate your UDFs to subsequent releases of your Cognos product, try to name your UDFs in such a way that there is little chance the names will be used by functions that are built into future versions of the product.

Supported Operating Systems

For information about supported operating systems and versions, visit the Cognos Support Web site (http://support.cognos.com).

UDF Procedures

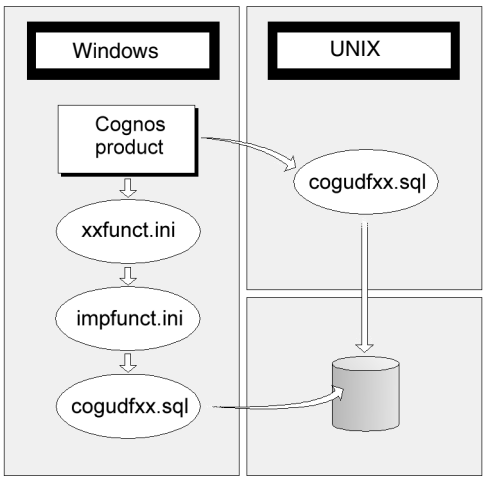
Use the following procedures to implement UDFs:

- Steps to Implement a Database UDF
- Steps to Use a Database UDF with Multiple Databases
- Steps to Implement an External UDF

The Appendix contains a list of the two-character database abbreviations used to form the names of the .sql and .ini files that you need to modify in these procedures.

Steps to Implement a Database UDF

The diagram below shows the files Cognos products use to access a database UDF:



1. Add the function to the database if the function does not already exist. (For more information, consult your RDBMS documentation.)
2. Add the function name to the .ini file for the database. By default, the file is in the *installation_location*\cer*n*\bin folder.
   The format of the filename is xxfunct.ini — a combination of the two-character identifier for the database and the letters ‘funct’. For example, the .ini file for Oracle is orfunct.ini.
   Insert the function name into the [Database-specific Function List] section in the .ini file.
3. Add the function name as a new section name to the end of the .ini file. The name should be in square brackets. For example, [MyFunction].
4. In the new section, add the definition of your function.

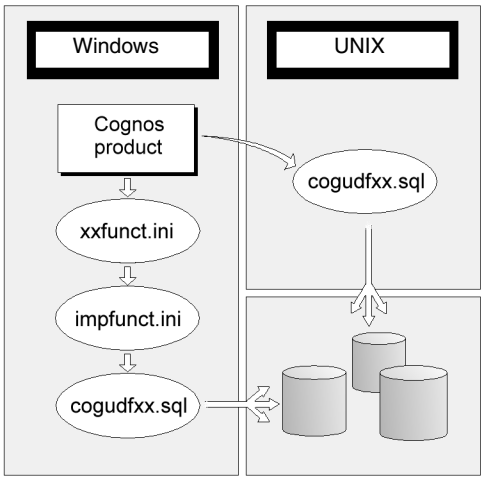
5. Add the function definition to the appropriate .sql file. By default, the file is in the *installation_location*\cer*n*\bin folder. The format of the filename is cogudfxx.sql — a combination of the letters 'cogudf' and the two-character identifier for the database. If there is no .sql file for the database in the Bin folder, create the file.

   You can use the Windows syntax-checking utility cogudf.exe (provided in the SDK) to verify what you have added to the .sql file. For more information, see the Appendix on (p. 35).

If the UDF has been added correctly to all the necessary files, the function will be visible in the expression editor when you start your Cognos product and connect to a database of the type using your UDF.

Steps to Use a Database UDF with Multiple Databases

The diagram below shows the files Cognos products uses to access a database UDF for more than one database:



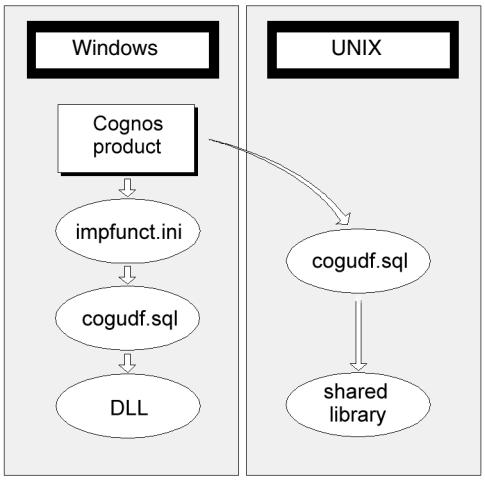
1. Add the function to the impfunct.ini file in the [Common Database Function List] section.
2. Add the function name as a new section name at the end of the impfunct.ini file. The name should be in square brackets. For example, [MyFunction].
3. Add the complete function definition under the new section name in the impfunct.ini file.
4. For each database that you want to use the function with, add the function name to the [Common Database Function List] section in the corresponding database .ini file (xxfunct.ini).
5. For each database that you want to use the function with, add the function definition to the corresponding .sql file (cogudfxx.sql) in the Bin folder. If there is no .sql file for a database you want to use with the UDF, create the file.

   Use the Windows syntax-checking utility cogudf.exe (provided in the SDK) to verify what you have added to the .sql files. For more information, see the Appendix on (p. 35).

If the UDF has been added correctly to all the necessary files, the function will be visible in the expression editor when you start your Cognos product and connect to a database of the types using your UDF.

Steps to Implement an External UDF

The diagram below shows the files Cognos products use to access an external UDF:



1. Create the DLL or shared library.
   The DLL should be copied to one of the following folders:
   - the Bin folder
   - one of the directories in the PATH environment variable
   - the Windows folder
   - the Windows system folder

   In UNIX, use one of the following environment variables to point to the shared library:
   - LD_LIBRARY_PATH (Solaris)
   - LIBPATH (AIX)
   - SHLIB_PATH (HP-UX)

   For more information, refer to the UNIX man pages for 'ld'.
2. Add the name of the external UDF to the [Built-in Function List] of the impfunct.ini file.
3. Add the function name as a new section name to the end of the impfunct.ini file. The name should be in square brackets. For example, [MyFunction].
4. Under the new section name, add the complete definition of your function.
5. Add the function definition to the cogudf.sql file.
   Use the Windows syntax-checking utility cogudf.exe (provided in the SDK) to verify what you have added to the .sql file. Before you copy .sql files to your UNIX environment, you can check them on a PC with this utility. For more information, see the Appendix on (p. 35).

If the UDF has been added correctly to all the necessary files, the function will be visible in the expression editor when you start your Cognos product.

Chapter 2: Modify the .ini Files

In this chapter, you will learn about

- the impfunct.ini file, and how to modify it
- the structure of the database-specific .ini files

Overview

As mentioned in the previous chapter, you need to modify .ini files to make your UDFs available to your Cognos product. By default, the .ini files are installed in *installation_location*\cer*n*\bin folder. The files you need to modify are

- impfunct.ini (for database UDFs that will be used with more than one database system, and for external UDFs)
- one or more database-specific .ini files (for database UDFs only)

For information on the list of database-specific .ini files, see the Appendix on (p. 35).

The Contents of impfunct.ini

The impfunct.ini contains definitions of

- common database functions
- database UDFs
- external UDFs

This file is divided into a number of sections. To create UDFs that are available to more than one database, you modify the section [Common Database Function List].

Note

If you create a database UDF, do not modify the existing entries in the section [Built-in Function List]. However, if you create an external UDF, add the new entry to this section.

Each function name is shown on a single line and is followed by an equals sign (=). For example,

```
A_function=
B_function=
```

The function names are not case-sensitive: A_function is the same as a_function.

Further down in the .ini file, there are sections named after these functions. These sections define what the functions actually do, as well as how they appear in the expression editor.

Function Definition Properties

Each function definition contains a number of properties. Each property occupies a single line. In the example below for the absolute function, the line starting with "tip=" has been wrapped for practical reasons in this document, but should be on a single line in the .ini file.

```
[absolute]
label=absolute
param=1
return=NM
1=NM;numeric_exp
exp=absolute ( ^1 )
tip=Syntax: absolute (numeric_exp) \nReturns the
absolute value of numeric_exp. The sign of negative
values is changed to positive. Examples: absolute (5)
returns 5; absolute (-5) returns 5.
tip1=Numeric expression
```

The label property is optional, but the others are required. Each property entry is limited to 4096 characters, except for the tip parameter. Its limit is 512 characters.

The following text describes the function properties:

Function Properties

label

Optional. The name of the function as it appears in the Function List of the expression editor. If no label is specified, the name of the function is used.

param

Specifies the number of parameters to the function. Must be less than 32767.

return

Specifies the type of value returned by the function. The value is one of these two-letter codes:

NM Numeric

CH Character

DA Date

DT Datetime

IN Interval

TM Time

In complex expressions (those that have differing return values, such as the Oracle decode function), the value is a single character in the range 'A' to 'Z' and denotes a parameter-class (see below). The return type will be the same as the type of the parameter-class. For example:

return=**B**

1=DA,DT,NM,CH,IN,TM:A;expression

2=DA,DT,NM,CH,IN,TM:A;search1

3=DA,DT,NM,CH,IN,TM:**B**;result1

4=DA,DT,NM,CH,IN,TM:A;search2

5=DA,DT,NM,CH,IN,TM:**B**;result2

6=DA,DT,NM,CH,IN,TM:**B**;default

Parameter-class A (parameters 1, 2 and 4) will have the same type, and parameter-class B (parameters 3, 5 and 6) will have the same type. The type of the return value in the example will be the type of parameter-class B. The Oracle decode function (of which this is an excerpt) could be called as in the following examples (where NM? denotes a numeric value, CH? denotes a character value and DA? denotes a date value):

decode (NM1, NM2, CH3, NM4, CH5, NM6, CH7, CH8) returns character

decode (DA1, DA2, NM3) returns numeric

<parameter description>

Each parameter is described by a comma-separated list of possible types followed by a descriptive string, separated by a semicolon (;).

There is one description property for each parameter and the properties are labeled 1, 2, 3 , and so on.

The types are:

NM Numeric

CH Character

DA Date

DT Datetime

IN Interval

TM Time

Some functions can be passed parameters of different types. To do this, list the different types, separated by commas, in the numbered parameter description. For example, the add-days() function accepts either a Date or a DateTime value as its first parameter:

1=DA,DT;date_exp

In complex expressions, the list of types for each parameter description can be followed by a colon (:) and a single character denoting a parameter-class. All parameters in a parameter-class must have the same list of types in their parameter description attributes.

All actual parameters in a parameter-class must be the same type.

exp

The expression. The SQL for the exp attribute must be valid. One or more space or tab characters separate each token.

Parameters are substituted into the generated SQL statement by replacing the parameter-markers with the SQL generated for each parameter. A parameter marker is a caret character (^) followed by a number. For example, ^1.

Complex expressions can be specified (for example, see the date-to-string entry in the impfunct.ini file), but most definitions consist of the function name followed by a comma-separated list of parameter-markers.

To reference an external UDF, the function name must be enclosed in square brackets ([…]):

```
exp=[myfunct] ( ^1 , ^2 )
```

Square brackets are also used to indicate optional sequences. In addition, a pair of square brackets followed by an asterisk (*) indicates a repeating sequence. For example, the decode function definition contains this expression definition with repeating and optional sequences:

```
exp=decode ( ^1 , ^2 , ^3 [ , ^4 , ^5 ]* [ , ^6 ] )
```

[, ^4, ^5]* is repeating, and [, ^6] is not repeating. You can have one repeating sequence and one optional sequence together in a function definition. You cannot however have more than one of either in the function definition. The repeating sequence (if present) must precede the optional sequence (if present).

For a repeating sequence, Cognos products processes all the parameters you provide until none is left.

Here is another example of how a repeating sequence works. If you define

```
exp=FooFunc( ^1 [, ^2 , ^3]*)
```

You can call the function as follows:

```
FooFunc(A, B, C)
FooFunc(A, B, C, D, E)
FooFunc(A, B, C, D, E, F, G)
```

tip

Specifies the fly-out text displayed for the function in the expression editor. The limit for the string is 512 characters.

parameter tip

Specifies the fly-out text for the parameters. There must be one parameter tip for each parameter. Parameter tips are labeled tip1, tip2, tip3, and so on.

type

Specifies that the function definition has a repeating parameter or a dynamic return type, or both. Its value must be a character string containing the character D or R, or DR:

- D — Return type is dynamic (can vary depending on parameter types)
- R — Parameters can be repeating.
- DR — Parameters can be repeating and the return type is dynamic.

Database-Specific .ini Files

The format of the filenames for the database-specific .ini files is xxfunct.ini — a combination of the two-character identifier for the database and the letters 'funct'. See the Appendix for a list of the two-character identifiers.

Database-specific .ini files are processed when a connection is made to the database.

[Built-in Function List]

This section has the same syntax as in the impfunct.ini file. If a function is listed here, it is implemented in the database as native SQL. Note that you add the external UDF definition to the [Built-in Function List] in the impfunct.ini file, not to the ones in any of the database-specific .ini files.

[Common Database Function List]

This section lists common database functions that are actually implemented by the database. If a function that is listed in the [Common Database Function List] in the impfunct.ini file is not listed here, then it is not implemented by the database and will not appear in the Function list in the expression editor.

[Database-specific Function List]

This section lists functions that are implemented in the corresponding database.

Individual Function Definitions

The format of this section is the same as in the impfunct.ini file. You can override exp and parameter description attributes in impfunct.ini by redefining them here. For example, you might want to override these attributes to change the exp to a database-specific syntax or modify the types of parameters that the function takes. You cannot override any of the other attributes.

The functions that are listed in the [Database-specific Function List] must have all their attributes specified in this section.

If a function is defined twice in this section, the last definition is used.

Chapter 3: Implement Database UDFs

In this chapter, you will learn how to

- modify the appropriate .sql files
- use the syntax for defining database UDFs
- define overloaded database UDFs

Database SQL Files (cogudfxx.sql)

To set up a database UDF you need to

- add the function to the database (if necessary)
  Refer to your RDBMS documentation for details.
- modify the .ini file for the database type
- modify the impfunct.ini file if the UDF is to be used with more than one database system
- add the function definition to the .sql file for the database type

A database UDF .sql file contains UDF definitions and can contain comments as well. You create comments using two dashes (- -). Any text following the dashes on the same line is ignored.

Note: The cogudfxx.sql files cannot contain any Cognos SQL data manipulation language (DML) statements, such as SELECT or INSERT.

Syntax for Defining a Database UDF

Items shown in bold are required. Items shown in square brackets [ ] are optional. Any of the set of items in braces { } may be present. The format and case of the database names you use must match the names you used to define the UDF in the database.

```
<database function declaration> ::=
  DECLARE [ DATABASE] [ <function type> ] FUNCTION
  <function name> [<formal parameter list>]
  RETURNS <data type>
  FUNCTION NAME <database function name> ;

<function type> ::=
      SCALAR

<function name> ::=
      [<logical database name>.]<identifier>

<formal parameter list> ::=
      ([<data type>] [{ , <data type>} … ] )

<returns data type> ::=
      <data type>

<database function name> ::=
    : [<catalog>.][<schema>.]<function name>

<logical database name> ::=
  : <identifier>

<catalog> ::=
  : <identifier>

<schema> ::=
  : <identifier>

<function name> ::=
  : <identifier>

<identifier> ::=
  : text
  | "<text>"

<data type> ::=
     STRING
   | BOOLEAN
   | NUMBER
   | BINARY
   | DATE
   | TIME
   | TIMESTAMP
   | INTERVAL
   | BLOB
   | TEXT
   | <literal value>

<literal value> ::=
  : '<text>'
```

Syntax Description

- A parameter may be of type BINARY, but the return value of a function cannot be BINARY.
- The initial <function name> is the name your Cognos product uses to identify the UDF.
- The STRING type represents both fixed and varying-length character values.
- The NUMBER type represents the following types of numeric values:
  - SMALLINT
  - INTEGER
  - DECIMAL
  - NUMERIC
  - REAL
  - FLOAT
  - DOUBLE PRECISION
- You can indicate that a database function has no parameters and requires no parentheses by not including parentheses for the parameter list in the function definition.
- The <database function name> is the name the underlying database uses to identify the function. This allows database functions defined in another schema or catalog to be referenced in SQL using a simple, user-defined name. This name is used exactly as you enter it — if any special characters or delimited names are required, the entire function name should be delimited by double quotation marks. Embedded double quotation marks are indicated by two successive double quotation marks.
- A parameter may have a fixed value that is specified by a text string delimited by single quotation marks. Embedded single quotation marks are indicated by two successive single quotation marks. A parameter with a specified, fixed value may not be combined with any other parameter data type.
- The return value of a database function may not be a literal value.

Examples of Database UDF Definitions

The .sql files contain definitions like these:

```
DECLARE FUNCTION substitute( STRING, STRING )
RETURNS STRING
FUNCTION NAME lion.tiger.subst;

DECLARE DATABASE FUNCTION localDB.timeOfDay()
RETURNS TIME
FUNCTION NAME sysfunc.time_of_day;

Declare Database Function "My Custom Function"
( Number, 'Fixed value here!' )
Returns Number
Function Name "My Catalog"."My Schema"."Custom Function";
```

Overloaded Database Functions

Cognos products can access database UDFs regardless of how they are implemented in the underlying database. But the functions must be accessible within an SQL statement.

Some databases, such as DB2 and Informix, support function overloading. Overloading means that two or more functions with the same name can exist. Overloaded functions are differentiated by the number and types of parameters that each function takes.

Cognos products do not currently support function overloading. Every function must have a unique name. To be able to access different versions of an overloaded function, Cognos products require that each instance of the function be uniquely identified in the appropriate .sql file for the database.

The syntax for identifying the function is the same whether the database UDF is written in C, Java, or a proprietary database procedural language such as Oracle PL/SQL.

Example

The function MYFUNC is defined twice in DB2. The first definition is:

```
CREATE FUNCTION myfunc (SMALLINT)
    RETURNS SMALLINT
    EXTERNAL NAME mylib!myfunc
    LANGUAGE C…
```

The second definition is:

```
CREATE FUNCTION myfunct (FLOAT, FLOAT)
    RETURNS FLOAT
    EXTERNAL NAME mylib!myfloatfunc
    LANGUAGE C…
```

To use these UDFs from within a Cognos SQL statement, you must define the two functions as follows in the cogudfd2.sql file:

```
DECLARE DATABASE FUNCTION myIntFunc (NUMBER)
RETURNS NUMBER
FUNCTION NAME myfunc;

DECLARE DATABASE FUNCTION myFloatFunc (NUMBER, NUMBER)
RETURNS NUMBER
FUNCTION NAME myfunc;
```

Oracle Functions

The example below is in C, but you can use PL/SQL to define a UDF in an Oracle database. For more information, refer to your Oracle documentation.

Example

```
CREATE FUNCTION obtain_city
( longitude IN FLOAT,
  latitude IN FLOAT )
RETURN FLOAT AS EXTERNAL
LIBRARY mylibrary
NAME "obtain"
LANGUAGE C;
```

The definition of this function in the cogudfor.sql file is:

```
DECLARE DATABASE FUNCTION obtainCity (NUMBER, NUMBER)
RETURNS STRING
FUNCTION NAME obtain_city;
```

DB2 Functions

As noted above, DB2 supports function overloading. A unique name must be used to identify each version of the overloaded function to your Cognos product.

Example

```
CREATE FUNCTION obtain_city (FLOAT, FLOAT)
RETURNS VARCHAR(50)
EXTERNAL NAME "mylibrary!obtain"
LANGUAGE C
PARAMETER STYLE DB2SQL
DETERMINISTIC
NO SQL
NO EXTERNAL ACTION
```

The function declaration in the cogudfd2.sql file is the same as for the Oracle example.

Informix Functions

Cognos products do not support Informix statement local variables (SLVs). An SLV is a variable within a stored procedure that you can then refer to within the calling SQL statement. For more information about SLVs, refer to your Informix documentation.

Informix supports function overloading. Again, a unique name must be used to identify each version of the overloaded function to your Cognos product.

Informix Universal Server UDFs can be written outside the database. UDFs for Informix Online are written as procedures. The following example is for Informix Universal Server 9.12.

Example

```
CREATE FUNCTION obtain_city (longitude FLOAT, latitude FLOAT)
RETURNING VARCHAR(50)
EXTERNAL NAME "/usr/lib/local/obtain.so"
LANGUAGE C
END FUNCTION;
```

The function declaration in the cogudfif.sql file is the same as for the Oracle example.

Chapter 4: Implement External UDFs

In this chapter, you will learn

- how to add your external UDFs to the cogudf.sql file
- how to code external UDFs
- about requirements and points to keep in mind when designing an external UDF library

Overview

To implement an external UDF, you create and deploy your shared library or DLL, then

- add the definitions of external UDFs to the cogudf.sql file
- add the function name and definition to the impfunct.ini file

This chapter discusses how to add an entry to the cogudf.sql file and how to design an external UDF library.

Requirements and Restrictions

The following requirements and restrictions apply to external UDFs:

- You can use a language other than C to build an external UDF, but the UDF must support the C calling convention.
- The name you give to an external UDF must be unique.
- An external UDF can take a maximum of 16 parameters.
- Parameters cannot assume more than one data type.
- Binary, text, and BLOBs are not supported as either parameters to, or return values from, external UDFs.
- All data items supplied to and returned by an external UDF are aligned, dependent upon the data type.
- All external UDFs must return a void value.

The External UDF SQL File (cogudf.sql)

By default, the cogudf.sql file is located in the *installation_location*\cer*n*\bin folder. Cognos products read this file when they encounter a function that they don't recognize as a database function. The cogudf.sql file tells your Cognos product

- where to find the UDF
- the UDF name in C
- the type of each parameter
- the return type of the UDF

Search Paths for .sql Files

In Windows environments, the .sql files are searched for in

1. the current working folder
2. the folder specified in the `servicesL` section of the Cognos.ini file
3. the Bin folder

In UNIX environments, the .sql files are searched for in

1. the current working directory
2. the directory pointed to by the `COGUDFSQL` environment variable
3. the directory pointed to by the `DMDBIMI` environment variable

Contents

The cogudf.sql file contains Cognos SQL data definition language (DDL) statements. It cannot contain any Cognos SQL data manipulation language (DML) statements, such as SELECT or INSERT, nor can it contain database UDF definitions.

As with the other .sql files, cogudf.sql can also contain comments. You create comments using two consecutive dashes (- -).

Syntax for an External UDF

In the syntax below, items shown in square brackets [ ] are optional. There may be zero or more occurences of the items in braces { }.

```
<external function declaration> ::=
  DECLARE EXTERNAL [ <function type> ]
  FUNCTION <function name> ( <formal parameter list> )
  RETURNS <data type> [ <result cast> ]
  FUNCTION NAME <external function name> ;

<function type> ::=
      SCALAR

<function name> ::=
      <identifier>

<formal parameter list> ::=
      [<data type>] [ { , <data type> } … ]

<result cast> ::=
      CAST AS <data type>

<external function name> ::=
      [ <module name> . ] <external function name>

<module name> ::=
      <text>

<external function name> ::=
      <text>

<identifier> ::=
  : <text>
  | "text"
<data type> ::=
      CHARACTER <optional length>
    | CHAR <optional length>
    | BINARY <optional length>
    | SMALLINT <optional scale>
    | INTEGER <optional scale>
    | QUADWORD <optional scale>
    | DECIMAL <precision and scale>
    | NUMERIC <precision and scale>
    | REAL
    | DOUBLE PRECISION
    | DATE
    | TIME
    | TIMESTAMP
    | INTERVAL

<optional length> ::=
   ( <length> )

<optional scale> ::=
   ( <scale> )

<precision and scale> ::=
   ( <precision> [, <scale> ] )

<length> ::=
   <unsigned integer>

<precision> ::=
   <unsigned integer>

<scale> ::=
   <signed integer>
```

Syntax Description

- <function name> is the name by which your Cognos product identifies the external function. As mentioned, the names of the functions must be unique — if a duplicate name is encountered, only the first function with that name will be recognized.
- CHARACTER and CHAR are synonyms.

- A parameter of type CHARACTER must not have a specified size. The size of all character parameters is determined by the input value to the function prior to its execution. If the return type of a function is CHARACTER, then a length must be specified. This is the maximum length that may be handled by the external UDF.
- A parameter may be of type BINARY, but the return or cast value of a function cannot be BINARY.
- The <module name> of the <external function name> is the name of the dynamic link library (DLL) or shared library containing the function. The <function name> of the <external function name> is the name of the function within the library. For example, myDLL.myFunction.
- The default scale of SMALLINT, INTEGER and QUADWORD values is zero. A scale of value greater than zero indicates the number of digits that must appear after the decimal point. A scale of value less than zero represents the number of additional digits which must appear prior to the decimal point. For example, if an INTEGER value of 123 had a scale of 2, its actual value would be 1.23. If its scale was -2, its actual value would be 12300.
- The default scale of DECIMAL and NUMERIC values is zero. DECIMAL and NUMERIC scale values are either zero or greater than zero.
- Cognos function names are not case sensitive. External function names are case sensitive.

Examples of External UDF Definitions

The .sql file contains definitions such as:

```
DECLARE EXTERNAL FUNCTION substitute( CHAR, CHAR )
SCALAR RETURNS CHAR(10) CAST AS CHAR(50)
FUNCTION NAME rxxfunc.subst;

DECLARE EXTERNAL FUNCTION timeOfDay()
RETURNS TIME
FUNCTION NAME sysfunc.time_of_day;
```

These two functions would be declared in C as shown below.

```
void subst( CogChar *, CogChar *, CogChar ** )
{    /* Code goes here. */
    return;

} /* subst */

void time_of_day( CogTime ** )
{    /* Code goes here. */
    return;

} /* time_of_day */
```

The above examples assume that the library can be found by the dynamic loader without the full path specification. The name of the DLL or shared library may either be a simple name or contain a path or file extension. When the name contains a path or file extension, the library name must be enclosed within double quotation marks. For example, the following is a valid DECLARE FUNCTION statement:

```
DECLARE EXTERNAL FUNCTION myFunction( INTEGER )
RETURNS INTEGER
FUNCTION NAME "/usr/local/lib/mylib.so.1".my_function;
```

Create the External UDF Library

Considerations for Windows Environments

All UDFs must be exported from the DLL in which they reside to be accessible to your Cognos product. It is possible for your Cognos product to simultaneously access one or more DLLs that each contain UDFs.

DLLs containing UDFs must be located in

- the current directory of the application to be executed
- one of the directories specified in the PATH environment variable
- the Windows directory
- the Windows system directory

All DLLs must be 32 bit and compiled with 8 byte alignment.

Considerations for UNIX Environments

UDFs may exist in any number of shared libraries. The UDF shared library must be located in a directory accessible to the run-time linker. You can point to a UDF shared library using the UNIX environment variables previously mentioned:

- LD_LIBRARY_PATH (Solaris)
- LIBPATH (AIX)
- SHLIB_PATH (HP-UX)

The cogudf.h File

The cogudf.h file should be included in every file that contains UDFs. This file provides access to all the necessary macros, type definitions and function definitions. The cogudf.h file includes cogudfty.h, so you do not need to explicitly include the latter.

If a library of UDFs uses any of the functions defined in cogudf.h, the library must be linked with the UDF library (udflib.lib in Windows, udfLib.a on UNIX).

An external UDF can use functions from a variety of libraries (static or shared) or DLLs. However, a UDF must release all allocated memory before it completes its task.

NULL Parameter Checking

All parameters to a UDF are passed by reference. A value of NULL for a particular parameter indicates that the associated value from the partially completed SQL query is itself NULL.

Because dereferencing a NULL parameter will cause your Cognos product to end abnormally, a UDF must check if any of its parameter values is NULL before doing further processing.

If the UDF encounters a NULL parameter value, it should return a value of NULL.

The following code shows how to check for NULL values:

```
Void myNewFunction( CogInt32 *  param1,
                    CogInt32 *  param2,
                    CogInt32 ** result );
{
    if( param1 == NULL || param2 == NULL )
    {
        *result = NULL;
    }
    else
    {
        **result = *param1 - *param2;
    }

    return;

} /* myNewFunction */
```

The return value of all UDFs is the final parameter of the function. It is always accessed by single indirection to set its value to NULL and by double indirection to assign it a value (as shown above).

A UDF may accept as many as 16 input parameters, plus the additional parameter for the return value.

Error Handling

There is currently no error handling for UDFs. If an error occurs, a UDF must be able to recover and return an appropriate value.

Data Types

The following fixed set of data types is available to UDFs:

Data Type	Description
CogChar *	A pointer to a null-terminated character string.
CogInt16	Signed 16-bit integer.
CogInt32	Signed 32-bit integer.
CogInt64	Signed 64-bit integer.
CogFloat32	32-bit floating point value.
CogFloat64	64-bit floating point value.
Packed Decimal (CogUInt8)	Fixed length, exact numeric value (described below).
CogDate	Binary representation of a year/month/day value.
CogTime	Binary representation of a hour:minute:second:millisecond value.
CogTimestamp	Binary representation of a date/time value.
CogInterval	Binary representation of a days hour:minute:second:millisecond value.
CogBinary	Fixed length binary data.

These data types (except for Packed Decimal) are defined in the C header file cogudfty.h.

The CogChar, CogInt16, CogInt32, CogFloat32 and CogFloat64 data types are represented by underlying C data types. Apart from the CogInt64 data type, the remaining data types are all represented as proprietary values. The CogInt64 data type is supported as a native 64 bit integer data type in all the operating systems supported by UDFs.

CogInt64 Native Types	
Windows	LONGLONG
Solaris	long long
HP-UX	long long
AIX	long long

Each UNIX platform requires a specific compiler option to enable support for 64-bit integers. Refer to your UNIX documentation for more information.

Packed Decimal Data Type

Packed decimal values are represented using a single nibble (4 bits) for each digit in a number. All values are padded with leading and trailing zeros to ensure that a fixed number of digits is always represented by a packed decimal value. To indicate whether the final (right most) nibble of a packed decimal is positive or negative, it is assigned one of the following values:

- COG_DECIMAL_POSITIVE_1
- COG_DECIMAL_POSITIVE_2
- COG_DECIMAL_NEGATIVE

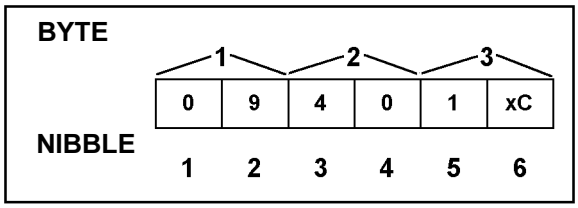
These are defined in cogudfty.h. Keep in mind that it is simpler for your UDF to check for the negative indicator than for the positive indicators.

The maximum precision of a packed decimal value is 77. The scale of a packed decimal value must be greater than or equal to zero and less than or equal to the precision.

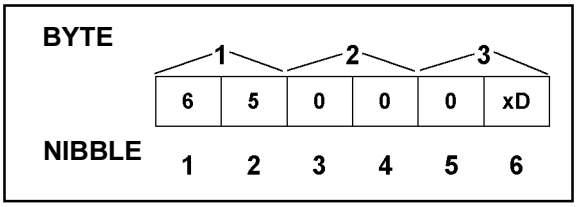
The number of bytes required to represent a numeric value depends on the precision of the data type and whether the precision is odd or even. The calculation for the number of bytes required to represent a packed decimal value is:

```
number of bytes = ( number of digits / 2 ) + 1
```

For example, if a parameter is defined with precision 4 and scale 2, the value 94.01 would be represented as:



If a parameter is defined with precision 5 and scale 3, the value -65 would be represented as:



Note in the above example that the value -650 cannot be represented in a decimal parameter with precision 5 and scale 3. When you define UDFs to accept or return decimal (numeric) values, ensure that the function will only be passed acceptable values or that the range of values accepted is large (large precision). If your application does not require more than 19 digits of accuracy, you can use CogInt64 values instead.

UDFs manipulate packed decimal values as an array of unsigned bytes, the number of which is determined by the number of digits in the packed decimal value (as described above).

The following two functions are provided with the UDF SDK to convert packed decimal values to and from character strings. The character strings must be allocated prior to calling the cogDecimalToString function long enough to hold the resulting value.

```
CogChar * cogDecimalToString(
    CogUInt8 * pDecimal,
    CogChar  * pString
    CogInt32   precision,
    CogInt32   scale,
);

CogUInt8 * cogStringToDecimal(
    CogChar  * pString,
    CogUInt8 * pDecimal
    CogInt32   precision,
    CogInt32   scale
```

);

The following is the definition of a UDF that accepts a packed decimal value as an input parameter and returns a 32-bit floating point value:

```
void myPDFunction( CogUInt8 * param1, CogFloat32 ** result )
{
    /*
     * The precision and scale are pre-defined for a UDF.  For
     * this example, assume the precision is five and the scale
     * is two.
     */

     if( param1 == NULL )
     {
        *result = NULL;
     }
     else
     {
        CogInt16   i;

        for( i = 0, **result = 0.0; i < 5; i++ )
        {
            **result *= 10;

            if( i & 1 )
            {
                **result += ( param1[ i / 2 ] ) & 0xF;
            }
            else
            {
                **result += ( param1[ i / 2 ] >> 4 ) & 0xF;
            }
        }

        /*
         * Account for the scale.
         */

        **result /= 100;

        if( ( param1[ 2 ] & 0x0F ) == COG_DECIMAL_NEGATIVE ){
            **result = -**result;
        }
     }

     return;

} /* myPDFunction */
```

Date Data Type

Date values are represented by the CogDate structure. The following functions are provided as part of the UDF SDK to manipulate date values:

```
CogDate * cogConstructDate(
    CogDate * pCogDate,
    Int32     year,
    Int32     month,
    Int32     day
);
CogInt32 cogGetDatePart(
    CogDate * pCogDate,
    UInt32    datePart
);
```

The possible values for the datePart parameter are:

- COG_DATE_YEAR
- COG_DATE_MONTH
- COG_DATE_DAY

The following example shows a UDF that adds a year to the supplied date:

```
void myDateFunction( CogDate * param1, CogDate ** result )
{
    CogInt32 year;
    CogInt32 month;
    CogInt32 day;

    if( param1 == NULL )
    {
        *result = NULL;
    }
    else
    {
        year  = cogGetDatePart( param1, COG_DATE_YEAR );
        month = cogGetDatePart( param1, COG_DATE_MONTH );
        day   = cogGetDatePart( param1, COG_DATE_DAY );

        year++;

        cogConstructDate( *result, year, month, day );
    }

    return;

} /* myDateFunction */
```

Time Data Type

Time values are represented by the CogUdfTime structure. The following functions are provided as part of the UDF SDK to manipulate time values:

```
CogTime * cogConstructTime(
    CogTime * pCogTime,
    Int32     hour,
    Int32     minutes,
    Int32     seconds,
    In32      milliseconds
);
CogInt32 cogGetTimePart(
    CogTime * pCogTime,
    UInt32    timePart
);
```

The possible values for the timePart parameter are:

- COG_TIME_HOUR
- COG_TIME_MINUTE
- COG_TIME_SECOND
- COG_TIME_MILLISECOND

The following example shows a UDF that rounds a time value to the nearest minute:

```
void myTimeFunction( CogTime * param1, CogTime ** result )
{
    CogInt32 hour;
    CogInt32 minute;
    CogInt32 seconds;
    CogInt32 fseconds;

    if( param1 == NULL )
    {
        *result = NULL;
    }
    else
    {
        hour     = cogGetTimePart( param1, COG_TIME_HOUR );
        minute   = cogGetTimePart( param1, COG_TIME_MINUTE );
        seconds  = cogGetTimePart( param1, COG_TIME_SECOND );
        fseconds = cogGetTimePart( param1,
                                   COG_TIME_MILLISECOND );

        /*
         * Round up/down to the closest minute.
         */

        if( fseconds >= 500 )
        {
            seconds++;
        }

        if( seconds >= 30 )
        {
            minute++;
        }

        if( minute == 60 )
        {
            minute = 0;
            hour++;
        }

        if( hour == 24 )
        {
            hour = 0;
        }

        fseconds = 0;
        seconds  = 0;

        cogConstructTime( *result,
                          hour,
                          minute,
                          seconds,
                          fseconds );
    }

    return;

} /* myTimeFunction */
```

Timestamp Data Type

Timestamp values are represented by the CogTimestamp structure:

```
typedef struct {
    CogDate date;
    CogTime time;
} CogTimestamp;
```

The various CogDate and CogTime functions can be used to construct the fields within a CogTimestamp structure or to extract the relevant portions from it.

Interval Data Type

Interval values are represented by the CogInterval structure. The following functions are provided as part of the UDF SDK to manipulate interval values:

```
CogInterval * cogConstructInterval(
    CogInterval * pCogInterval,
    Int32         days,
    Int32         hour,
    Int32         minutes,
    Int32         seconds,
    In32          milliseconds
);
CogInt32 cogGetIntervalPart(
    CogInterval * pCogInterval,
    UInt32        intervalPart
);
```

The possible values for the intervalPart parameter are:

- COG_INTERVAL_DAYS
- COG_INTERVAL_HOUR
- COG_INTERVAL_MINUTE
- COG_INTERVAL_SECOND
- COG_INTERVAL_MILLISECOND

An interval specifies a number of days and a time value. An interval does not specify a particular date or time, but instead specifies an interval of time. For example, an interval could be the amount of time between two specific timestamp values.

Binary Data Type

Binary values are represented in the C language with a descriptor:

```
typedef struct
{
    Int32      length;
    CogUInt8 * data;
}
CogBinary;
```

The length field represents the number of bytes of binary data pointed to by the data field.

A function that returns a binary value has a pre-defined maximum size that is specified in the UDF definition file. A UDF must assign to the length field the number of bytes that contain valid data (if the value is not NULL).

Important Points about Data Types

When designing a UDF, you need to take into consideration the environment in which your UDF will be used. Some of the important questions to ask are:

- If the UDF returns a character string, what is the maximum size that it will support?
- What range of values should the UDF support?
- Should the UDF support a wide variety of data types (requiring implicit or explicit type casts)?

If you don't resolve these questions, your UDF may fail, cause system instability, or possibly run to completion but return either invalid or incorrect data.

Character Data Type

All character string parameters are supplied to a UDF as null-terminated strings. There is no restriction on the length of an input parameter to a UDF. However, the same is not true for the return value. The maximum length (excluding the null terminator) must be pre-defined in the cogudf.sql file. Cognos products provide the UDF with a buffer of the specified size (plus a byte for the null terminator). The result cannot be larger than the specified size.

The maximum length of a character string is 2^{31} - 1 bytes.

Also keep in mind that in non-English locales, the number of characters in a string does not necessarily equal the number of bytes. A character may require anywhere from 1 to 4 bytes. The size specified in the cogudf.sql file represents the maximum number of bytes, not characters.

Small Integer, Integer and 64-bit Integer Data Types

These exact numeric data types represent an increasing range of values. If the values to be handled by a parameter or return value always fall within a specific range, it is most efficient to choose the smallest applicable type.

Integer types are efficient for the purpose of performing arithmetic calculations and providing exact values. However, their range of values can be restrictive. Although 64-bit integers provide 19 digits of precision, they are not natively supported by all hardware.

Also note that an optional scale can be defined in the cogudf.sql file for an integer parameter that defines the number of digits that appear after the decimal point. This can be useful for monetary calculations.

Packed Decimal Data Type

As mentioned before, all packed decimal values, whether parameters or return values, must have a pre-defined precision and scale. Because of this, their usefulness to a generalized UDF may be limited. One possible solution is to use the maximum precision in all packed decimal parameters and return values.

Another problem with packed decimals is that there is no native support for them in C. All such values must be converted to another data representation before they can be manipulated.

Float and Double Precision Data Types

Float and double precision numeric values support a larger range of values than exact numeric data types, but only provide 7 and 15 digits of precision (for float and double, respectively). If you determine that you don't need greater precision, these approximate numeric data types are probably the best option for numeric parameters and return types.

Note that floating-point calculations can take longer to process than equivalent operations performed on exact numeric types, even if the computer in question has a floating point processor.

Binary Data Type

When dealing with binary data, it is assumed that the UDF has knowledge of all binary data supplied as a parameter, and that an application has knowledge of the contents of binary data returned from a UDF.

Implicit Casts

Type casts allow a function to be called with values that do not exactly match the data types of the function's parameters. Several implicit type casts are possible. If a cast cannot be performed, an error will occur during the preparation of an SQL query.

Although conversion from exact numeric types to approximate data types is supported, it is possible that significant digits may be lost in the conversion. Overflows may occur in conversions to and from packed decimal values, depending upon the precision of the packed decimal value.

The following table codifies the data types for easier representation in the second table. The last of the three tables provides a legend for the second table.

Data Type	Code	Data Type	Code
CogUInt16	UI16	CogChar	CH
CogInt16	I16	CogDate	DT
CogUInt32	UI32	CogTime	TM
CogInt32	I32	CogTimestamp	TS
CogInt64	I64	CogInterval	IN
CogFloat32	F32	CogBinary	BN
CogFloat64	F64	Packed Decimal	PD

To	I16	UI16	I32	UI32	I64	PD	F32	F64	CH	DT	TM	TS	IN	BN
From														
I16	+	+	+	+	+	+	+	+	+					
UI16	●	+	+	+	+	+	+	+	+					
I32	●	●	+	+	+	+	+	+	+					
UI32	●	●	●	+	+	+	+	+	+					
I64	●	●	●	●	+	+	+	+	+					
PD	●	●	●	●	●	+	+	+	+					
F32	●	●	●	●	●	●	+	+	+					
F64	●	●	●	●	●	●	●	+	+					
CH	+	+	+	+	+	+	+	+	+	+	+	+	+	
DT									+	+		+		
TM									+		+	+		
TS									+	+	+	+		
IN									+				+	
BN														+

Symbol	Legend
+	The cast is supported.
Blank	The cast is NOT supported.
●	Although the cast is supported, an overflow may occur.

Appendix

In this appendix, you will find

- a table of Cognos identifiers for supported database systems and the database-specific .ini and .sql filenames
- troubleshooting tips
- information about the cogudf.exe syntax-checking utility
- a list of the files that make up the UDF Software Developer's Kit

Database Identifiers

Database	Identifier	.ini File	SQL File
DB2	D2	d2funct.ini	cogudfd2.sql
Informix	IF	iffunct.ini	cogudfif.sql
Microsoft SQL Server	MS	msfunct.ini	cogudfms.sql
ODBC	OD	odfunct.ini	cogudfod.sql
Oracle	OR	orfunct.ini	cogudfor.sql
Sybase CT-Library	CT	ctfunct.ini	cogudfct.sql

Troubleshooting

This section provides corrective actions to take if you encounter the error messages or problems listed here.

Error

```
The function <function name> is not available as an external, database or built-in
function.
```

Corrective Action

Ensure that the .sql file is located in one of the following:

- the current working directory
- in Windows, the folder specified in the `ServicesL` section of the Cognos.ini file, or the Bin folder
- in UNIX, the directory specified by either the `COGUDFSQL` or `DMDBINI` environment variable

If the file appears to be located in the correct location, ensure that the function definition has no syntax errors. Check the syntax with the cogudf utility.

Error

```
External function <function name> requires 2 parameter(s) but 3 supplied.
```

Corrective Action

The incorrect number of parameters has been supplied to a function. Ensure that

- your function definition is correct
- it is invoked correctly in your SQL statement
- there is only a single definition of the function in the appropriate .sql file — only the first definition of a function will be used

Error

```
External function <function name> is not accessible.
```

Corrective Action

The function cannot be located. Possible explanations are:

- the name of the library as specified in the .sql file does not match the name of the shared library that contains the external function
- the name of the function in the C module and the C function name as defined in the .sql file are not the same
- in UNIX, the platform-specific library path environment variable has not been given a value
- in UNIX, the shared library is not in one of the directories specified in the library path environment variable
- in Windows, the shared library is not in one of the directories specified in the PATH environment variable
- the function has not been exported from the shared library or DLL
- the name of the function as specified in the export list does not match the name of the function in the C module

Garbled Data or Unexpected Exit

In Windows, if you receive garbled data from a function, or the function causes the application to abort, ensure that you have compiled your DLL with 8 byte alignment.

Database Errors

If you receive a database error message when invoking a database UDF within an SQL statement, ensure that

- the name of the UDF has been typed correctly in the corresponding .sql file
- the database UDF has been provided the necessary qualification

  For example, if the UDF is installed in a schema other than your default schema, the UDF will likely require a schema qualification in the function definition in the .sql file. In general, it is best to fully qualify database UDF function names to remove any ambiguity and to avoid database errors that may arise from the database being unable to locate the UDF.

The cogudf Syntax-Checking Utility

You can use the Windows utility cogudf.exe to help you test your .sql files for syntax errors.

If you are developing UDFs for a UNIX platform, you can use cogudf to debug your .sql files in a Windows environment before deploying the UDFs to your UNIX system(s).

To use cogudf, simply run it. It takes no command-line options. The cogudf utility first searches the current folder, then the folder specified in the ServicesL section of the Cognos.ini file.

The cogudf utility reports any errors that it finds in the .sql files. In Windows environments, the output is sent to the log file specified by the UDF Log File entry in the [ServicesL] section of the Cer*n*.ini. file:

```
[ServicesL]
UDF Log File=d:\temp\udf.log
```

If this entry does not exist or does not have a value, no information is written to a log file. If the entry exists, the file will be created, or appended to if an existing file is found.

Your Cognos product also writes to the log file when it encounters a UDF in an SQL expression.

In the UNIX environment, the environment variable COGUDFLOG points to the log file. If you specify this variable, the log file will only be written to when Request Server encounters a UDF.

Files in the UDF Software Developer's Kit

The SDK files and samples were created with Microsoft Developer Studio 97 with Visual C++ 4.2.

File	Description
cogudfty.h	For UNIX and Windows. A header file that defines the data types supported by UDFs. Included by cogudf.h.
cogudf.h	For UNIX and Windows. A header file that defines all necessary macros, types and functions for UDF functionality.
udfLib.a	For UNIX. A library containing the functions defined in cogudf.h. Must be linked to any DLL containing UDFs if any of the Cognos UDF support functions are used.
udflib.lib	For Windows. A library containing the functions defined in cogudf.h. Must be linked to any DLL containing UDFs if any of the Cognos UDF support functions are used.
cogudf.exe	For Windows. A utility that checks all available .SQL files containing database and external function definitions for syntax or semantic errors.
cogudf.sql	For UNIX and Windows. The .SQL file that contains the UDF definitions.
Database udf readme.txt	Readme file for the sample database UDF. Explains how to use the following files: Oracle UDF.txt Orfunct Update.txt Informix UDF.txt Cogudfxx.sql
Encrypt readme.txt	Readme file for the sample external encryption UDFs. Explains how to use the following files: UDFs. Impfunct Update.txt Encrypt.c Encrypt.def Encrypt.dll Cogudf Encrypt Update.sql Impfunct Encrypt Update.txt

File	Description
Html readme.txt	Files for the sample external HTML UDFs. Pubhtml.c Pubhtml.c Pubhtml.def Pubhtml.dll Cogudf HTML Update.sql Impfunct HTML Update.txt
Udf.mdp Udf.mak Udf.ncb	The Microsoft Developer Studio project files for the samples.

Index

V

W