



Net.Data 語言環境手冊

目錄

注意事項	v
商標	v
第1章 介紹	1
誰應該閱讀本書	1
關於本書中的範例	1
深入學習 Net.Data	1
第2章 使用語言環境	3
Net.Data 提供的語言環境	3
Java Applet 語言環境	4
Java 語言環境	8
ODBC 語言環境	9
Oracle 語言環境	9
PERL 語言環境	10
REXX 語言環境	10
SQL 語言環境	11
Sybase 語言環境	11
系統語言環境	11
第3章 架構一個語言環境	13
第4章 建立新的語言環境	15
語言環境程式設計	15
我應該提供哪一種語言環境介面？	15
處理輸入參數	15
處理使用者要求	16
處理輸出參數	16
通信錯誤狀況	16
第5章 語言環境介面	17
語言環境介面中的結構	17
dtw_lei.	17
dtw_parm_data	18
語言環境介面中的函數	19
常式 dtw_initialize()	20
常式 dtw_execute()	20
常式 dtw_getNextRow().	20
常式 dtw_cleanup()	20
語言環境公用程式函數	21
dtw_free().	23
dtw_getvar()	24
dtw_malloc().	25
dtw_row_SetCols().	26
dtw_row_SetV()	27
dtw_setvar()	28
dtw_strdup()	29
dtw_table_AppendRow().	30
dtw_table_Cols()	31
dtw_table_Delete().	32

dtw_table_DeleteCol()	33
dtw_table_DeleteRow()	34
dtw_table_GetN()	35
dtw_table_GetV()	36
dtw_table_InsertCol()	37
dtw_table_InsertRow()	38
dtw_table_MaxRows()	39
dtw_table_New()	40
dtw_table_QueryColnoNj()	41
dtw_table_Rows()	42
dtw_table_SetCols()	43
dtw_table_SetN()	44
dtw_table_SetV()	45
附錄. 語言環境模版	47
索引	59

注意事項

本出版品提及 IBM 產品、程式或服務時，不暗示 IBM 會在有業務的所有國家發行所有這些項目。凡提及 IBM 產品、程式或服務項目時，亦不表示只可用 IBM 的產品、程式或服務項目。受限於 IBM 的有效智慧財產權或其它合法的權利，任何功能上相等的產品、程式或服務，都可以用來代替 IBM 產品、程式或服務。除了由 IBM 設計的部份外，使用其它產品的評詁與驗證明確指定是使用者的責任。在本文件中包含著 IBM 所擁有之專利或暫准專利。而提供本文件，並不表示允許您使用這些權利。您可以書面方式提出特許權限之相關問題，並郵寄至：

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

獲得本程式的授權者，如需其相關資訊作為下列用途：(1) 在獨立創作的程式與其他程式（包括本程式在內）之間交換資訊(2) 互相使用彼此交換的資訊，請洽：

IBM Corporation
555 Bailey Avenue, W92/H3
P.O. Box 49023
San Jose, CA 95161-9023

上述資料之取得有其特殊要件，在某些情況下必須付費方得使用。

商標

下列詞彙是 IBM 公司在美國或其它國家或此兩者的商標：

AIX	OS/2
DB2	OS/390
IBM	OS/400

下列詞彙是以下其它公司的商標：

UNIX 是透過 X/Open Company Limited 獨家授權，在美國以及其它國家的註冊商標。

Microsoft、Windows、Windows NT 以及 Windows 95 標誌是 Microsoft Corporation 的商標或註冊商標。

其它公司、產品以及服務名稱可能以兩個星號（**）標示，表示是其它公司的商標或服務標示。

第1章 介紹

誰應該閱讀本書

本書討論 Net.Data 的語言環境，供您在呼叫程式或函數，或資料來源如 DB2、Oracle 或 Sybase 資料庫時使用。撰寫 Net.Data 巨集的人可使用這個資訊來找出 Net.Data 提供給您的語言環境能力。本書還含有打算自己撰寫語言環境當作 Net.Data 後端之使用者所需的資訊。您必須熟悉 C 程式設計語言以及*Net.Data* 程式設計手冊 與*Net.Data*參考手冊中的資訊。

關於本書中的範例

本書中使用的範例力求簡單以便展示特定的概念，因此並未考慮所有可能的情況。有些範例只是片段，無法單獨使用。

深入學習 Net.Data

請從以下的全球資訊網站取得其它的可用詳細資訊：

<http://www.software.ibm.com/data/net.data>

您可以取得 Net.Data 範例巨集、展示程式本書的最新版本、常問的問題的列示,以及在論壇上提出問題。

第2章 使用語言環境

Net.Data 在設計上容許以插入的方式加入新的語言以及資料庫介面。這些語言環境是當作 DLL 或共用程式庫的方式存取。DLL 的名稱架構在 Net.Data 起始設定檔中，並和一個語言環境名稱關聯。每一種語言環境必須支援一組由 Net.Data 定義的標準介面。當第一次對指定該語言環境的 FUNCTION 區塊進行函數呼叫時，Net.Data 便會載入起始設定檔中指定的 DLL。

Net.Data 會解析 Net.Data 巨集、維護 Net.Data 變數、提供與語言環境之間的介面、以及根據 REPORT 及 MESSAGE 區塊規格將輸入格式化。語言環境支援針對 Net.Data 定義的介面、使 Net.Data 參數可被語言處理器透過與語言相關的方式被存取、呼叫語言直譯器、以及以某種與語言相關的方式從語言直譯器接收傳回的變數。

Net.Data 提供的語言環境

Net.Data 提供多種語言環境，但不是所有的平台都支援所有的環境。第3頁的表 1 列出 IBM 提供的語言環境。想知道您的平台是否支援某一個語言環境，請參閱*Net.Data*參考手冊的附錄。

REPORT 與 MESSAGE 區塊可用於所有的 FUNCTION 區塊。這些區塊會被 Net.Data 處理，而不是語言環境。

RETURN_CODE 會在每一次對語言環境進行函數呼叫時，被 Net.Data 設定為函數傳回值。您可以檢查這個值來判斷函數呼叫是成功或失敗。Net.Data 使用 RETURN_CODE 值來處理對函數指定的任何區域或整體 MESSAGE 區塊。為了能夠和「DB2 WWW 連接」相容性，SQL_CODE 還會被設成相同的值。

這個表格顯現 Net.Data 提供的語言環境。在某些平台上，起始設定檔中的 ENVIRONMENT 陳述式也會被當作 Net.Data 的內建函數。請參閱 Net.Data README 檔或程式目錄，來取得您平台使用之語言環境陳述式的詳細資訊。

表 1. Net.Data 語言環境

語言環境	環境陳述式	說明
Java Applet	DTW_APPLET	Java applet 語言環境供您在您的 Net.Data 應用程式中使用 Java applet。要產生 applet 標籤，您必須提供 applet 標籤的限定元以及 applet 的參數列示。
預設 C	DTW_DEFAULT	在某些平台上，起始設定檔含有一個 Net.Data 內建函數需要的 ENVIRONMENT 陳述式。請參閱 <i>Net.Data</i> 參考手冊 來取得如何使用內建函數的詳細資訊。
純本文檔介面	DTW_FFI	有些平台上可使用純本文檔介面（FFI），供您使用純本文檔（或純文字檔）當作資料來源。有些平台在起始設定檔中使用 ENVIRONMENT 陳述式來支援 FFI。請參閱 <i>Net.Data</i> 程式設計手冊 來取得如何使用 FFI 的詳細資訊。
Java 應用程式	DTW_JAVAPPS	Net.Data 透過 Java 語言環境來支援您現有的 Java 應用程式。
ODBC	DTW_ODBC	ODBC** 語言環境透過 ODBC 介面來執行 SQL 陳述式，可存取多種資料庫管理系統。
Oracle	DTW_ORA	Oracle** 語言環境供您直接和 Oracle 資料通信。

表 1. *Net.Data* 語言環境 (繼續)

語言環境	環境陳述式	說明
PERL	DTW_PERL	PERL 語言環境可解譯 <i>Net.Data</i> 巨集 FUNCTION 區塊中指定的內部 PERL script，也可以執行儲存於個別檔案中的外部 PERL script。
REXX	DTW_REXX	REXX 語言環境可解譯 <i>Net.Data</i> 巨集 FUNCTION 區塊中指定的內部 REXX 程式，也可以執行儲存於個別檔案中的外部 REXX 程式。
SQL	DTW_SQL	SQL 語言環境透過 DB2 來執行 SQL 陳述式。SQL 陳述式的結果可用表格變數傳回。
Sybase**	DTW_SYB	Sybase** 語言環境供您直接和 Sybase 資料通信。
系統	DTW_SYSTEM	系統函數可支援 FUNCTION 區塊中 EXEC 陳述式識別的外部程式呼叫。系統語言環境以傳遞程式名稱參數給作業系統供執行的方式，來解譯 EXEC 陳述式。
Web 登錄	DTW_WEBREG	<i>Net.Data</i> Web 登錄可在某些平台上使用，可為應用程式相關的資料提供永久的儲存體。有些平台在起始設定檔中使用 ENVIRONMENT 陳述式來支援 Web 登錄。請參閱 <i>Net.Data</i> 程式設計手冊 來取得如何使用 Web 登錄的詳細資訊。

每一個語言環境都要求起始設定檔中含有一個 ENVIRONMENT 陳述式，伺服器的 /lib 或 /d11 目錄中含有一個共用程式庫或 dll 檔。建議您在進行變更之前，先備份起始設定檔。

Java Applet 語言環境

Java applet 語言環境供您在 *Net.Data* 應用程式中使用標準的 Java applet。

要產生 applet 標籤，您必需要指定 applet 標籤的限定元（在本範例中為 code、codebase、width、height）以及 applet 的參數列示。

下列段落說明可產生 applet 標籤的函數呼叫語法，以及 applet 標籤限定元參數的語法。Java applet 的使用範例是 第6頁的。

產生 Applet 標籤

呼叫 Java applet 的函數呼叫的語法如下：

```
@DTWA_AppletName(parameter1, parameter2, ..., parameterN)
```

DTWA_ 字首識別產生 applet 標籤的函數呼叫。*AppletName* 是產生標籤的 applet 的名稱。這個函數呼叫通常在 HTML 段落中進行。

Applet 標籤

由於 HTML 標籤的任何地方經常改變，*Net.Data* 擁有更富彈性但更強而有力的方法來設定值。您可以在 *Net.Data* 巨集中隨意指定 applet 標籤限定元。所有格式為 *AppletName.qualifier* 的變數都會被代換成 applet 標籤的限定元。以下是定義 applet 限定元的範例：

```
%define AppletName.Qualifier = "value"
```

在 `DEFINE` 段落中不需要實際指定值。您可以使用 `DTW_ASSIGN` 函數來設定值，也可以在 `SQL` 區段中自動設定。如果您未對 `AppletName.code` 變數定義一個變數，有一個預設的 `code` 參數會被加到 `applet` 標籤。 `code` 參數的值是 `AppletName.class`，其中 `AppletName` 是您 `applet` 的名稱。

Applet 標籤參數

當您在 `Net.Data` 中呼叫 `applet` 時，便會設定參數列示。您可以傳遞

- 簡單的 `Net.Data` 變數（包括 `LIST` 變數）
- 表格
- 表格直欄

永遠被產生的參數

每一次都會產生名稱爲 `DTW_NUMBER_OF_TABLES` 的標籤。值便是函數中使用之表格變數的個數。如果函數呼叫中不指定表格變數，這個標籤產生如下：

```
<param name = "DTW_NUMBER_OF_TABLES" value = "0" >
```

`DTW_APPLET_ALTTEXT` 變數可在最後一個 `PARAM` 標籤與終止 `APPLET` 標籤之間插入 `HTML`。這個變數含有的 `HTML` 本文，會顯示在不支援 `APPLET` 標籤的瀏覽器中。

您可以使用正規 `Net.Data` 變數來當作參數。如果您在 `DTWA_AppletName` 函數呼叫中指定一個變數，將產生一個具有與變數相同之名稱以及值的 `PARAM` 標籤。

也可以使用一個或多個表格變數當作參數。如果您在 `DTWA_AppletName` 函數呼叫中指定一個 `Net.DATA` 表格變數，將產生多個額外的 `PARAM` 標籤。首先，將對表格產生一個 `PARAM` 標籤，例如：

```
<param name = 'DTW_TABLE_i_NAME' value = "tname" >
```

變數 *i* 是表格的數目， *tname* 是表格的名稱。

接著，會產生 `PARAM` 標籤，來指定表格的橫列及直欄數，例如：

```
<param name = 'DTW_tname_NUMBER_OF_ROWS' value = "rows" >  
<param name = 'DTW_tname_NUMBER_OF_COLUMNS' value = "cols" >
```

表格的名稱是 *tname*，*rows* 是表格的橫列數，*cols* 是表格的直欄數。對於函數呼叫中指定的每一個唯一的表格，都將產生這一對標籤。

然後，將對表格中每一個直欄產生一個 `PARAM` 標籤，例如：

```
<param name = 'DTW_tname_COLUMN_NAME_j' value = "cname" >
```

表格名稱是 *tname*，*j* 是直欄數目，*cname* 是表格中的直欄名稱。

最後，將在每一個橫列中產生含有值的 `PARAM` 標籤：

```
<param name = 'DTW_tname_cname_VALUE_k' value = "val" >
```

表格名稱是 *tname*，*cname* 是直欄名稱，*k* 是橫列數目，*val* 是符合對應橫列與直欄之值的值。

當作參數的表格直欄

您可以在函數呼叫中，將表格直欄當作參數來傳遞。表格直欄參數的格式如下：

```
@DTWA_AppletName(DTW_COLUMN( x )Table)
```

其中 `x` 是 表格中的直欄名稱或直欄數。

對於直欄參數及表格，會傳遞相同的參數，`DTW_tname_NUMBER_OF_COLUMNS` 參數永遠設定為 1。只有直欄專用的參數標籤會被產生，而不會對格中的其它直欄產生參數標籤。

假設巨集檔中指定如下：

```
%define{
DATABASE = "celdial"
DTW_APPLET_ALTTEXT = "<P>對不起，您的瀏覽器不支援 Java。"
DTW_HTML_TABLE = "yes"
DTW_SAVE_TABLE_IN = "MyTable"
MyGraph.codebase = "'MyMachine.ibm.com'"
MyGraph.height = "200"
MyGraph.width = "400"
MyTitle = "這是我的標題"
%}
%SQL(A){
select age, count(age)
as ages from guests
where age >= 35
group by age
%}
%HTML_REPORT{
%EXEC_SQL(A)
@DTWA_MyGraph( MyTitle, DTW_COLUMN(ages) MyTable )
%}
```

在 `Net.Data` 巨集範例中，位於定義區段中的這些行會產生 `applet` 標籤的第一行：

```
MyGraph.codebase = "'MyMachine.ibm.com'"
MyGraph.height = "200"
MyGraph.width = "400"
```

`applet` 含有這些限定元：

```
<applet code = 'MyGraph.class'
codebase = 'MyMachine.ibm.com' width = '400'
height = '200' >
```

`Net.Data` 巨集檔 `SQL` 區段中的 `SQL` 查詢的結果，由結果表格 `MyTable` 傳回，這個表格由定義區段中的 `DTW_SAVE_TABLE_IN = MyTable` 所設定對巨集 `@DTWA_MyGraph(MyTitle, DTW_COLUMN(ages) MyTable)` 中的 `applet` 進行呼叫會產生完整的 `applet` 標籤，其中含有結果表格的相關資訊，如傳回的直欄個數、橫列個數、以及結果橫列。會對結果表格中的每一個儲存格，產生一個參數標籤，如下所示：

```
param name = 'DTW_MyTable_ages_VALUE_1' value = "4">
```

參數名稱 `DTW_MyTable_ages_VALUE_1` 參照表格 `MyTable` 中的表格儲存格（橫列 1，直欄 'ages'），它的值是 4。在對 `applet` 進行的呼叫中，關鍵字 `DTW_COLUMN` 提定：您只對結果表格 `MyTable` 中之直欄 `ages` 有興趣，如下所示：

```
@DTWA_MyGraph( MyTitle, DTW_COLUMN(ages) MyTable )
```

對範例產生的完整 `applet` 標籤看起來如下：

```

<applet code = 'MyGraph.class'
codebase = 'MyMachine.ibm.com' width = '400' height = '200' >
<param name = 'MyTitle' value = "這是我的標題" >
<param name = 'DTW_NUMBER_OF_TABLES' value = "1" >
<param name = 'DTW_TABLE_1_NAME' value = "MyTable" >
<param name = 'DTW_MyTable_NUMBER_OF_ROWS' value = "5" >
<param name = 'DTW_MyTable_NUMBER_OF_COLUMNS' value = "1" >
<param name = 'DTW_MyTable_COLUMN_NAME_1' value = "ages" >
<param name = 'DTW_MyTable_ages_VALUE_1' value = "4">
<param name = 'DTW_MyTable_ages_VALUE_2' value = "1">
<param name = 'DTW_MyTable_ages_VALUE_3' value = "1" >
<param name = 'DTW_MyTable_ages_VALUE_4' value = "1" >
<param name = 'DTW_MyTable_ages_VALUE_5' value = "1" >
<P>對不起，您的瀏覽器不支援 Java。<BR>
</applet>

```

Applet 類別建議

建議您從 DTW_Applet 類別來衍生 applet 的次類別，因為它的介面用於協助處理 APPLLET 標籤非常有用。Net.Data 提供以下這些介面：

- **int GetNumberOfTables()** 傳回 applet 標籤中找到的表格數目。
- **String [] GetTableNames()** 傳回 applet 標籤中找到的表格名稱的列示。
- **int GetNumberOfColumns(String table_name)** 傳回表格中的直欄數。
- **int GetNumberOfRows(String table_name)** 傳回表格中的橫列數。
- **String[] GetColumnNames(String table_name)** 傳回表格中之直欄的名稱。
- **String[][] GetTable(String table_name)** 傳回含有表格之橫列與直欄值的二維陣列字串。

這個範例顯示的 applet 類別，乃是衍生自使用這些介面的 DTW_Applet 類別之次類別：

```

import java.io.*;
import java.applet.Applet;

public class myDriver extends DTW_Applet
{
    public void init()
    {
        super.init();

        if (GetNumberOfTables() > 0)
        {
            String [] tables = GetTableNames();
            printTables(tables);
        }
    }

    private void printTables(String[] tables)
    {
        String table_name;

        for (int i = 0; i < tables.length; i++)
        {
            table_name = tables[i];
            printTable(table_name);
        }
    }

    private void printTable(String table_name)
    {
        int nrows = GetNumberOfRows(table_name);
        int ncols = GetNumberOfColumns(table_name);
    }
}

```

```

        System.out.println("表格： " + table_name + " 含有 " + ncols + " 個直欄與
" +
                                nrows + " 個橫列。");

        String [] col_names = GetColumnNames(table_name);

        System.out.println("-----");

        for (int i = 0; i < ncols; i++)
            System.out.print(" " + col_names[i] + " ");
        System.out.println("\n-----");

        String [][] mytable = GetTable(table_name);

        for (int j = 0; j < nrows; j++)
        {
            for (int i = 0; i < ncols; i++)
                System.out.print(" " + mytable[i][j] + " ");

            System.out.println("\n");
        }
    }
}

```

Java 語言環境

Net.Data 透過 Java 語言環境來支援您現有的 Java 應用程式。

Net.Data 擁有不斷成長的 Java Database Connectivity (JDBC**) 支援；這一版支援 Java applets 與 Java 方法（或應用程式），可透過 JDBC API 來存取 DB2。

可從下列站台取得 JDBC 的詳細資料：

- IBM 軟體（IBM Software）備有 JDK 1.0，在您使用 JDBC 與 Net.Data 時需要它：
<http://www.software.ibm.com/data/db2/jdbc/>
- JavaSoft 備有額外的 JDBC 驅動常式、JDBC API 文件、以及 JDBC 的最新更新資料：
<http://splash.javasoft.com/jdbc/>

撰寫 JDBC 應用程式或 applet 和撰寫使用 DB2 CLI 或 ODBC 來存取資料庫的 C 應用程式非常相似。應用程式與 applet 之間的主要差別是：應用程式可能需要特定的軟體如 DB2 Client Application Enabler，才能夠和 DB2 通信。 applet 則依賴可使用 Java 的 Web 瀏覽器，從屬站上不必安裝 DB2 程式碼。

您的系統需要完成一些架構，才能夠使用 JDBC。這些注意事項在 DB2 JDBC 應用程式與 Applet 支援（DB2 JDBC Application and Applet Support）中有討論：

<http://www.software.ibm.com/data/db2/jdbc/db2java.html>

要從 Java applet 使用 JDBC，請在您的 Web 伺服器啟動 DB2 的 JDBC applet 伺服器。要從 Java 方法使用 JDBC：

- 在起始設定檔中，將您要使用的 client 的名稱加到 DTW_JAVA 環境陳述式中。
- 在伺服器上啟動 JAVA。
- Net.Data 巨集中的函數名稱必須和 Java 函數名稱一致。

ODBC 語言環境

Open Database Connectivity (ODBC) 語言環境可透過 ODBC 介面來執行 SQL 陳述式。ODBC 的基礎是 X/Open SQL CAE 規格，可讓單一的應用程式存取多個資料庫管理系統。

要使用 ODBC 語言環境，您必須具備 ODBC 驅動常式以及驅動常式管理程式。您的 ODBC 驅動常式文件含有如何安裝以及架構 ODBC 環境的說明。

在 ODBC 環境下傳送 SQL 陳述式的方式和其它的 `Net.Data` 函數相似。這個範例是一個 `Net.Data` 巨集，可將多個 SQL 陳述式傳送給 Oracle** 資料庫。使用 `DATABASE` 變數的平台必須指定和 `ODBC.INI` 檔中之資料來源相同的資料庫。

```
%DEFINE {
    DATABASE="qeora7"
    SHOWSQL="YES"
    table="int_null"
    LOGIN="netdata1"
    PASSWORD="ibmdb2"
}%

%function(dtw_odbc) oracle1() {
create table int_null (int1 int, int2 int)
}%

%function(dtw_odbc) oracle2() {
insert into $(table) (int1) values (111)
}%

%function(DTW_ODBC) oracle3() {
insert into $(table) (int2) values (222)
}%

%function(dtw_odbc) oracle4() {
select * from $(table)
}%

%function(DTW_ODBC) oracle5() {
drop table $(table)
}%

%HTML(REPORT) {
@oracle1()
@oracle2()
@oracle3()
@oracle4()
}%
```

Oracle 語言環境

Oracle 語言環境供您直接和您的 Oracle 資料通信。您的伺服器上必須安裝 Oracle 的 SQL*Net。有關詳細資訊，請參閱：

http://www.oracle.com/products/networking/html/stdn_sqlnet.html

這個語言環境不支援儲存程序。`DATABASE` 變數不必存取 Oracle 資料庫。

在起始設定檔中定義好 Oracle 語言環境加上執行 SQL*Net 後，您便可以將您的資料送出到 Web。使用函數來呼叫的 Oracle 語言環境的範例如下：


```
%FUNCTION(DTW_ORA) STL1() {
insert into ${table} (int1,int2) values (111,NULL)
%}
```

PERL 語言環境

PERL 語言環境可解譯您在 Net.Data 巨集之 FUNCTION 區塊中指定的插入式 (inline) PERL script,它也可以處理儲存在伺服器上之個別檔中的外部 PERL script。外部 PERL script 的呼叫性是由 FUNCTION 區塊中之 EXEC 陳述式加以識別，範例如下：

```
%EXEC{ perl-script-name [optional parameters] %}
```

PERL 語言環境無法直接傳遞或取回 Net.Data 變數，必須透過下列方式，PERL script 才可存取它們：

- 使用 *putenv()* 函數，將輸入參數當作環境參數傳遞給 PERL script，以及使用 PERL *\$ENV{}* 函數來取回。
- 以寫入指名管路的方式（它的名稱是在環境變數 DTWPIPE 中傳遞），將輸出參數傳回給語言環境。寫入指名管路的資料的格式是 *name = "value"*，和在 DEFINE 陳述式上的相同。如果有多個資料項目寫入管路，必須使用新行或空白字元分隔每一個項目。利用這種方法寫入時，如果寫入時輸出參數有對應的變數名稱，則新值會取代現行值。如果寫入的變數名稱沒有對應的輸出參數，則它會被忽略。

以下的 Net.Data Net.Data 巨集範例顯現 如何執行此動作。

```
%FUNCTION(DTW_PERL) today() RETURNS(result) {
    $date = 'date';
    chop $date;
    open(DTW, "> $ENV{DTWPIPE}") || die "無法開啓: $!";
    print DTW "result = \"$date\"\n";
%}
%HTML(INPUT) {
    @today()
%}
```

相同的函數可改寫成下一個範例，假設 PERL script 是存放在名為 today.prl 的外部檔案：

```
%FUNCTION(DTW_PERL) today() RETURNS(result) {
    %EXEC { today.prl %}
%}
```

PERL 語言環境程式可透過它們的 Net.Data 名稱來存取表格參數的值。表格 T 的直欄標題是 T_N_i，欄位值是 T_V_i_j。

REPORT 與 MESSAGE 區塊可用於任何 FUNCTION 區段中。它們會被 Net.Data 處理而不是語言環境。不過，PERL 程式可以將本文寫出到標準輸出裝置資料流及直接操作輸出 HTML 表格頁面。

REXX 語言環境

REXX 語言環境可以解譯 Net.Data 巨集之 FUNCTION 區塊中指定的插入式 (inline) REXX 程式，也可以執行儲存在個別檔案中外部的 REXX 程式。外部 REXX 程式的呼叫是由 FUNCTION 區塊中的陳述式加以識別，例如：

```
%EXEC{ REXX-program-file-name [optional parameters] %}
```


REXX 語言環境使用 `RexxStart()` API 來告訴 REXX 直譯器執行指定的檔案，接著將後面跟有檔案名稱的參數傳遞給程式，就像您生前在命令行上輸入的一般。對於 REXX 程式，所有的參數都是透過 `ARG[1]` 來接收。

只有在 FUNCTION 區塊的可執行陳述式區段中才會執行變數替代。不過，REXX 程式可存取參數，不論程式是定義在內部的 FUNCTION 區塊或外部的個別檔案。REXX 語言環境使用 REXX 語言處理器的 `RexxVariablePool()` 函數來和 REXX 程式共用 `Net.Data` 變數。這樣可容許 REXX 程式直接處理參數列示中識別的 `Net.Data` 變數。

REXX 程式可存取表格參數的值並當作 REXX 主變數 (stem variable)。對於 REXX 程式，表格 `T` 的直欄標題是 `T_N.i`，欄位值是 `T_V.i.j`。

SQL 語言環境

SQL 語言環境用於透過 DB2 來執行 SQL 陳述式。SQL 陳述式的結果可用 `Net.Data` 預設表格或您指定的表格來傳回。

`Net.Data` 支援您授權的任何 SQL 陳述式。當您呼叫 `Net.Data` 當作 `cgi` 應用程式時，您可以就每一個 HTML 段落和一個資料庫連接，但您必須使用 `DATABASE` 變數指定資料庫名稱 (OS/390 除外)。如果您的 DB2 資料庫和 Web 伺服器位於同一個機器上，您不必進行其它設定。否則，視您使用的平台而定，您可以使用 `Client Application Enabler (CAE)` 來存取遠端資料庫，也可以使用 `Database Connection Services (DDCS)` 來取得 DB2 支援的所有異動支援。您還可以使用 `DataJoiner` 來存取其它資料庫。使用 `DataJoiner`，您可以對支援的資料庫使用二階段確認寫入 (two-phase commit)。

對於使用連結檔的平台，起始設定檔必須指定 `BIND_FILE` 變數，範例如下：

```
BIND_FILE      ./DTWSQL.BND;
```

Sybase 語言環境

Sybase 語言環境供您直接和您的 Sybase 資料通信。伺服器上要安裝 Sybase 的 `Open Client CONNECT`。有關詳細資訊，請參閱：

<http://www.sybase.com/products/entcon/openccl.html>

Sybase 語言環境不支援大型物件如影像或音效。只有對不具有 `SELECT` 陳述式的程序支援儲存程序。

您可以使用函數來呼叫 Sybase 語言環境，範例如下：

```
%function(DTW_SYB) STL1() {  
  insert into $(table) (int1,int2) values (111,NULL)  
  %}
```

系統語言環境

系統語言環境支援 FUNCTION 區塊中 `EXEC` 陳述式識別的外部程式呼叫。

語言環境以傳遞程式名稱與參數給作業系統使用 C 語言 `system()` 函數呼叫處理的方式，來解譯 `EXEC` 陳述式。這個方法不容許像 REXX 語言環境那樣，讓外部程式直接傳遞或取回 `Net.Data` 變數，因此必須以下列方式處理變數：

- 使用 `putenv()` 函數將輸入參數當作環境變數傳遞給外部程式，並可由外部程式取回：

- Unix CSHELL script，以環境變數名稱開頭加上錢號（\$）的方式參照環境變數，例如 \$x.
- PERL 語言 script，以參照關聯陣列 ENV 的方式來參照它們，例如 %ENV{'x'}.
- DOS 批次檔，以使用百分比符號（%）來括住的方式來參照變數名稱，例如 %x%.
- 大部份的平台，是以寫入以環境變數 DTWPIPE 來傳遞名稱之指名管路的方式，將輸出參數傳回給語言環境。（(Net.Data for OS/400 使用環境變數將參數傳回給語言環境。）寫入指名管路的資料的格式是 *name="value"*，和在 DEFINE 陳述式上的相同。如果有多個資料項目寫入管路，必須使用新行或空白字元分隔每一個項目。如果變數名稱有對應的輸出參數，新的值會置換現行值。沒有對應輸出參數的變數名稱會被忽略。

系統語言環境程式可透過 Net.Data 名稱來存取表格參數的值。表格 T 的直欄標題是 T_N_i，欄位值是 T_V_i_j。

第3章 架構一個語言環境

Net.Data 起始設定檔 DB2WWW.INI 中已經事先架構多個語言環境。起始設定檔中每一個語言環境擁有一個 ENVIRONMENT 陳述式，其中含有該語言環境專用的資訊。ENVIRONMENT 陳述式的格式如下：

```
ENVIRONMENT(  
  type) library-name([用法  
  parameter, ...])
```

以下範例顯現 Net.Data 提供之語言環境的 ENVIRONMENT 陳述式。這些範例示範您可指定的所有參數。您放入 ENVIRONMENT 陳述式的變數，就是您容許 Net.Data 巨集寫出器在巨集中寫入或置換的變數。起始設定檔中的語言環境陳述式也用於 Net.Data 內建函數。請參閱*Net.Data 參考手冊* 的附錄或 Net.Data README 檔來取得與平台相關的資訊，或請參閱程式目錄來取得其它的範例。

```
ENVIRONMENT (DTW_SQL)      DTWSQL      ( IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM)  
ENVIRONMENT (DTW_SYB)      DTWSYB      ( IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM)  
ENVIRONMENT (DTW_ODBC)     DTWODBC     ( IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION_SCOPE, SHOWSQL, ALIGN)  
ENVIRONMENT (DTW_APPLET)   DTWJAVA     ( )  
ENVIRONMENT (DTW_JAVAPPS)  ( OUT RETURN_CODE ) CLIETTE "DTW_JAVAPPS"  
ENVIRONMENT (DTW_PERL)     DTWPERL     ( OUT RETURN_CODE )  
ENVIRONMENT (DTW_REXX)     DTWREXX     ( OUT RETURN_CODE )  
ENVIRONMENT (DTW_SYSTEM)   DTWSYS      ( OUT RETURN_CODE )
```

OS/390 的 SQL 與 ODBC 存取稍有不同：

```
ENVIRONMENT (DTW_SQL)      DTWSQL      ( IN LOCATION, DB2SSID, DB2PLAN,  
TRANSACTION_SCOPE, ALIGN)  
  
ENVIRONMENT (DTW_ODBC)     DTWODBC     ( IN LOCATION, TRANSACTION_SCOPE, ALIGN)
```

這些都是您必須對每一個語言環境指定的參數：

- 一個語言環境類型。

這是使這個語言環境和 Net.Data 中定義之 FUNCTION 區塊互相關聯的名稱。您必須在 FUNCTION 區塊定義中指定類型來告訴 Net.Data，在函數呼叫中，由這個語言環境來處理呼叫。

- 一個 DLL、共用程式庫或服務程式的名稱。

這是含有供 Net.Data 呼叫之語言環境介面的物件的名稱。在 OS/2 上，指定的 DLL 名稱不必附加 .dll 副檔名。在 AIX 上，指共用物件的名稱，指定時附加 .o 副檔名。在 OS/400，指定的服務程式名稱要附加 .SRVPGM 副檔名。OS/390 的 DLL 檔沒有附檔名。請查看您平台之 Net.Data 附隨的起始設定檔，來取得如何指定這個名稱的資訊。請考慮使用完整的路徑名稱，來確定 Net.Data 可以找到 DLL 或共用程式庫。

- 一個參數列示。

這個列示中的參數以及 FUNCTION 區塊定義中指定的參數，會在每一個函數呼叫時傳遞給語言環境。它們是由在 FUNCTION 區塊定義中指定的 dtw-lei-t 結構(其後跟有參數的)parm_data_array 欄位 加以來傳遞，您必須先在 Net.Data 巨集中將這些參數定義為變數，之後才可以進行函數呼叫。FUNCTION 區塊中的參數名稱，必須和區塊外的引數一致。任何這些參數在被函數修改時，會保有函數完成時修改過的值。

當 Net.Data 啟動時架構資訊即被讀入，但語言環境 DLL 或共用程式庫則要到識別語言環境的 FUNCTION 區塊被呼叫時，才會被載入。DLL 會保持載入的狀態，直到 Net.Data 結束為止。

每一個 ENVIRONMENT 陳述式必須由一行構成。語言環境可以使用 *dtw_getvar()* 常式來接收架構變數的值，以及使用 *dtw_setvar()* 常式來修改變數的值。請參閱 第24頁的『*dtw_getvar()*』以及 第28頁的『*dtw_setvar()*』。

語言環境可能需要額外的架構資訊。您可以透過架構變數陳述式，在 Net.Data 起始設定檔中指定此資訊。這些參數的格式如下：

NAME [=] 值字串

例如：

DB2INSTANCE = DB2

第4章 建立新的語言環境

在您決定建立新的語言環境之前，請先判斷附隨 Net.Data 之 IBM 提供的語言環境是否符合您的要求。如果您決定建立新的語言環境，則您必須先完成下列二個步驟：

1. 建立一個 DLL 或共用程式庫，其中含有您要提供之語言環境介面常式的集合。必須提供 `dtw_execute()` 介面，而且所有提供的介面必須完全符合 `dtwle.h` C 語言表頭中定義的原型。

另外，所有的介面必須可在 DLL 或共用程式庫之外使用，以便 Net.Data 可以呼叫它們。

2. 決定您的 ENVIRONMENT 架構陳述式，接著將它加到 Net.Data 起始設定檔中。請參閱 第13頁的『第3章 架構一個語言環境』來取得詳細資訊。

新增 ENVIRONMENT 架構陳述式到起始設定檔的工作，只需簡單的編輯作業便可完成。不過，首先要建立 DLL 或共用程式庫則較為複雜。

語言環境程式設計

您必須先撰寫語言環境，才能夠建立。請參閱 第47頁的『附錄. 語言環境模版』來取得語言環境原始檔案的範例。當您撰寫您要的語言環境時，必須考慮許多程式設計上的問題。以下數個段落說明較常見的問題。

我應該提供哪一種語言環境介面？

請使用 第47頁的『附錄. 語言環境模版』顯現的模版，如果您要自己撰寫語言環境來存取 Net.Data 的服務程式的話。

如果您的語言環境需要配置資源或起始設定資料，或者有任何處理和巨集呼叫的範圍（scope）有關的話，請考慮提供 `dtw_initialize()` 介面。這個介面在第一次對您的語言環境進行呼叫時，在每一個巨集呼叫中被呼叫的次數不會超過一次。如果沒有對您的語言環境進行任何函數呼叫，則 `dtw_initialize()` 介面不會被呼叫。如果您提供 `dtw_initialize()` 介面，則您還可以選擇提供 `dtw_cleanup()` 介面。這個介面在每一個巨集呼叫時也只會被呼叫一次。它只會在 `dtw_initialize()` 介面被呼叫時才會被呼叫，並且是當作巨集呼叫之終止的一部份被呼叫。

`dtw_getNextRow()` 介面非常特殊，而且只在當作資料庫語言環境的一部份，或者可一次處理一個橫列之資料的語言環境時才較為有用。

處理輸入參數

輸入參數永遠會傳遞給 `dtw_execute()` 介面當作 `dtw_lei_t` 結構的一部份，該結構是唯一傳遞給 `dtw_execute()` 的參數。參數是依照它們在 Net.Data 巨集的 FUNCTION 區塊定義中的指定順序傳遞，起始設定檔中指定的隱含參數以及隨後的明示參數也會被傳遞。它們是透過 `dtw_lei_t` 結構的 `parm_data_array` 欄位傳遞，每一個參數都是在各自的 `dtw_parm_data_t` 結構中。程式模版中的 `processInputParms()` 常式顯示一種處理輸入參數的可能方式。

處理使用者要求

語言環境處理使用者要求的方式，由該要求的收到方式決定。您有多種方式可用來進行要求和語言環境之間的溝通：

- 透過 FUNCTION 區塊中指定的函數名稱。指定的函數名稱會在每次進行函數呼叫時，透過 `dtw_lei_t` 結構的 `function_name` 欄位遞給語言環境。
- 透過 FUNCTION 區塊參數列示。在參數列示中指定位置上的參數可能會指定使用者要求。參數會在每一次函數呼叫時透過 `dtw_lei_t` 結構的 `parm_data_array` 欄位傳遞給語言環境。
- 透過 FUNCTION 區塊的可執行陳述式段落。指定的可執行陳述式會在每次函數呼叫時，透過 `dtw_lei_t` 結構的 `exec_statement` 欄位傳遞給語言環境。

處理輸出參數

這完全要由您的語言環境以及它如何處理使用要求來決定。不過，一旦語言環境有它希望傳回給 `Net.Data` 巨集的資料，它便可以修改由 `dtw_lei_t` 結構 `parm_data_array` 欄位傳遞之參數值的方式進行。程式模版中的 `processOutputParms()` 常式顯示一種處理輸出參數的可能方式，以及多個如何設定字串與表格參數值的範例。

通信錯誤狀況

函數呼叫的成功與失敗，可透過隱含 `Net.Data` 巨集參數 `RETURN_CODE` 來通信。這個變數會在從對 `dtw_execute()` 介面進行的呼叫傳回後，由 `Net.Data` 設定。它的值會被設定為 `dtw_execute()` 呼叫本身的傳回值。 `Net.Data` 接著會使用這個值來處理 `Net.Data` 巨集 `MESSAGE` 區塊，如果曾對這個函數呼叫指定一個的話。

如果您未曾指定 `MESSAGE` 區塊，或在指定的 `MESSAGE` 區塊中沒有用來處理 `dtw_execute()` 傳回值的項目， `Net.Data` 將會顯示 `dtw_lei_t` 結構的 `default_error_message` 欄位的內容。這個欄位在 `dtw_execute()` 常式中任何時候都可被語言環境設定。程式模版中的 `setErrorMessage()` 常式顯示如何設定 `default_error_message` 欄位的範例。

第5章 語言環境介面

您可以透過這些常式來存取 `Net.Data` 的服務程式：

- `dtw_initialize()`
- `dtw_execute()`
- `dtw_getNextRow()`
- `dtw_cleanup()`

`dtw_execute()` 是唯一必須由語言環境提供的常式。

當 `Net.Data` 被呼叫來處理 `Net.Data` 巨集,而它發現有對使用語言環境的函數發出呼叫時,將執行下列步驟：

1. 呼叫 `dtw_initialize()`(如果它存在語言環境中而且未曾被呼叫過的話)。
2. 呼叫 `dtw_execute()`。
3. 呼叫 `dtw_getNextRow()` (如果從 `dtw_execute()` 成功傳回時, `dtw_execute()` 指示 `dtw_getNextRow()` 應該被呼叫)。

`Net.Data` 會傳遞一個語言環境介面結構 (`dtw_lei`) 至它呼叫的語言環境常式。這個結構含有一個內含要傳遞給語言環境常式之參數列示的參數資料陣列,以及其它內容。被 `Net.Data` 呼叫的常式會處理要求、更新參數資料陣列中的參數(如果有的話),並傳回給 `Net.Data`。`Net.Data` 接著會處理參數資料陣列、更新它的參數副本來反映被語言環境常式設定的新值,然後繼續處理 `Net.Data` 巨集。一旦 `Net.Data` 巨集處理完成後,`Net.Data` 會對所有的語言環境呼叫 `dtw_cleanup()` (如果它適用於語言環境的話),然後返回到 Web 伺服器。

下列一節說明 `Net.Data` 語言環境介面,包括結構以及各種語言環境介面常式如何與 `Net.Data` 與交互作用。

語言環境介面中的結構

`Net.Data` 使用二種結構和語言環境溝通：`dtw_lei` 與 `dtw_parm_data`。

`dtw_lei`

每一個語言環境的介面常式會收到一個指向 `dtw_lei` 結構的指標：

```
typedef struct dtw_lei {
    char *function_name;          /* 語言環境介面          */
    int  flags;                   /* 函數區塊名稱          */
    char *exec_statement;         /* 語言環境介面旗號      */
    dtw_parm_data_t *parm_data_array; /* 語言環境陳述式        */
    char *default_error_message; /* 參數陣列               */
    void *le_opaque_data;        /* 預設訊息               */
    void *row;                   /* 語言環境專用資料      */
    char reserved[64];           /* 用於一次一橫列處理*/
} dtw_lei_t;                    /* 保留                   */
```

`dtw_lei_t` 的欄位將在稍後討論。`function_name` 欄位含有指向含有函數區塊之名稱的字串的指標。這在語言環境顯示錯誤訊息時非常有用。

旗號欄位供 Net.Data 用來和語言環境通信。旗號欄位是以使用下列常數來執行 OR 運算的方式來加以設定：

- DTW_STMT_EXEC 會被 Net.Data 設定，來對語言環境的 dtw_execute() 常式進行指示：目前傳遞的語言環境陳述式（在 exec_statement 欄位）是在 EXEC 陳述式中。
- DTW_END_ABNORMAL 會被 Net.Data 設定，來對語言環境的 dtw_cleanup() 常式進行指示：發生異常或非預期的狀況，以及語言環境應該在 Net.Data 結束之前執行任何必要的清除（即釋放保留的資源）。
- DTW_LE_FATAL_ERROR 會被語言環境介面常式設定，來對 Net.Data 進行指示：語言環境中已經發生嚴重錯誤。如果這個旗號為已設定，Net.Data 會停止處理 Net.Data 巨集、呼叫所有作用中之語言環境的 dtw_cleanup() 函數同時將旗號設定為 DTW_END_ABNORMAL、以及列印預設訊息，然後跳出。只有在語言環境呼叫傳回非零值時，旗號才會被檢查。
- DTW_LE_MSG_KEEP 會被語言環境介面常式設定，以便向 Net.Data 指示：由 default_error_message 指向的儲存體不應該被釋放。如果未設定，Net.Data 將試圖釋放儲存體。
- DTW_LE_CONTINUE 會被語言環境的 dtw_execute() 常式設定，以便向 Net.Data 指示：dtw_getNextRow() 應該被呼叫。只有在該旗號被設定的情況而且呼叫 dtw_execute() 常式的傳回值是零的狀況下，Net.Data 才會呼叫 dtw_getNextRow()。

exec_statement 欄位含有指向含有 FUNCTION 區塊中的可執行陳述式（在變數替代之後）的指標，或 EXEC 陳述式中的檔案名稱與參數。

parm_data_array 欄位含有指向 dtw_parm_data 結構之陣列的指標。陣列以含有零的 parm_data 結構終止。dtw_parm_data 結構供 Net.Data 用來傳遞參數與關聯的值給語言環境，以及取回由語言環境對變數值進行的任何變更。有關詳細資訊，請參閱以下段落。

default_error_message 欄位可被語言環境設定為一個描述錯誤狀況的字元字串。如果從語言環境介面函數傳回時，函數呼叫的傳回值不是零而且傳回值與訊息區塊中的訊息值不相符時，將會顯示預設訊息。否則，將會顯示從訊息區塊選取的訊息。

le_opaque_data 欄位可設定為任何 language_environment 的介面常式。Net.Data 將儲存指標並將它傳遞給它呼叫之語言環境的介面常式。在處理 Web 巨集後以及在傳回到呼叫者 Net.Data 之前，指標會設定為 NULL。由於欄位是執行緒專用，所以語言環境可以儲存與執行緒相關的資料。如果您有 dtw_cleanup() 常式，請使用這個欄位，這樣常式便可以釋放與 le_opaque_data 欄位相關的儲存體。

橫列欄位會在呼叫語言環境的 dtw_getNextRow() 函數之前被 Net.Data 設定為一個橫列物件。dtw_getNextRow() 會使用 Net.Data 橫列公用程式常式在物件中插入一個表格資料橫列。Net.Data 接著處理橫列，並持續呼叫 dtw_getNextRow() 直到沒有要處理的橫列為止。

保留欄位供 IBM 使用。

dtw_parm_data

dtw_parm_data 結構由 Net.Data 用來傳遞參數給語言環境。參數有三種來源：

- FUNCTION 區塊定義中指定的明示參數
- Net.Data 起始設定檔中 ENVIRONMENT 架構陳述式指定的參數
- FUNCTION 區塊定義中 RETURNS 關鍵字中指定的傳回變數

首先會傳遞明示參數，接著是 ENVIRONMENT 陳述式中指定的參數，接著是傳回變數。

dtw_parm_data 結構的格式如下：

```
typedef struct dtw_parm_data {          /* 參數資料          */
    int   parm_descriptor;              /* 參數描述子          */
    char *parm_name;                    /* 參數名稱            */
    char *parm_value;                   /* 參數值              */
    void *res1;                         /* 保留                */
    void *res2;                         /* 保留                */
} dtw_parm_data_t;
```

parm_descriptor 欄位描述參數的類型及使用。Net.Data 使用下列常數執行 OR 運算，來設定這個欄位。

- DTW_IN 由 Net.Data 設定，指定某參數是一個輸入專用（input-only）的參數。
- DTW_OUT 由 Net.Data 設定，指定某參數是一個輸出專用（output-only）的參數。
- DTW_INOUT 由 Net.Data 設定，指定某參數是輸入與輸出兩用的參數。
- DTW_STRING 由 Net.Data 設定，指示參數值是指向字串的指標。
- DTW_TABLE 由 Net.Data 設定，指示參數值是指向表格的指標。

Net.Data 永遠會將 parm_descriptor 欄位設定為 DTW_IN、DTW_OUT 或 DTW_INOUT，並和 DTW_STRING 與 DTW_TABLE 進行邏輯運算。

parm_name 欄位是指向含有參數名稱之字串的指標。如果參數是一個文字字串，這個指標會被 Net.Data 設定為空值（null）。

parm_value 欄位是指向含有參數值之物件的指標。如果參數的值不是已經定義的值，這個指標會被 Net.Data 設定空值（null）。

res1 與 res2 是保留欄位。

parm_name 與 parm_value 都是指向從 Net.Data 執行堆集配置之物件的指標。如果 parm_name 或 parm_value 被置換為其它字串，原始字串必須被釋放同時置換為指向從 Net.Data 堆集配置的字元字串的指標。請使用 dtw_malloc() 及 dtw_free() 來完成此動作。

語言環境介面中的函數

一個語言環境必須具備一個 dtw_execute() 常式。其它常式則是選用性的。如果 Net.Data 巨集參照沒有 dtw_execute() 常式的語言環境，Net.Data 將傳回錯誤訊息同時停止處理 Net.Data 巨集。

語言環境必須在被處理以供呼叫的 Net.Data 巨集中被參照。語言環境常式的呼叫順序必須如下：

1. dtw_initialize()
2. dtw_execute()
3. dtw_getNextRow()
4. dtw_cleanup()

以下各節說明這些常式。

常式 dtw_initialize()

在 Net.Data 要求的處理範圍中，Net.Data 會只呼叫一次一個語言環境的 dtw_initialize() 函數（如果已經定義），即呼叫第一次參照該語言環境被的函數區塊。後續的語言環境參照會略過 dtw_initialize() 函數的呼叫。

這個函數不會影響訊息區塊處理。正值或零的回覆碼表示繼續處理；負值回覆碼表示不繼續處理。如果回覆碼為非零的值而且 default_error_message 欄位中定義有預設的訊息，將發出預設訊息；如果沒有預設訊息，Net.Data 將發出錯誤訊息。

常式 dtw_execute()

每當 Net.Data 巨集處理一個 FUNCTION 區塊時，dtw_execute() 語言環境介面函數都會被呼叫。當 dtw_execute() 函數完成時，續發事件由 dtw_execute() 函數是否設定 dtw_lei 結構中的 DTW_LE_CONTINUE 旗號決定。

如果 DTW_LE_CONTINUE 旗號為已設定，Net.Data 會執行下列步驟：

1. 處理 dtw_execute() 函數之傳回值的訊息區塊
2. 呼叫語言環境的 dtw_getNextRow() 函數
3. 處理報告區塊
4. 處理 dtw_getNextRow() 函數之傳回值的訊息區塊
5. 如果 dtw_getNextRow() 已經設定 DTW_LE_CONTINUE 旗號，將以步驟 2 的 dtw_getNextRow() 函數繼續進行處理。否則，一次一橫列（row-at-a-time）處理會終止，Net.Data 將繼續處理 Net.Data 巨集。

當 dtw_execute() 函數進行必須用來減少報告區塊處理之所有處理時，執行效能通常會被最佳化。例如，SQL 語言環境的 dtw_execute 常式可產生報告區塊階段期間要處理的整個表格。

常式 dtw_getNextRow()

當符合例這些條件時，會呼叫 dtw_getNextRow() 語言環境介面函數：

- 呼叫語言環境的 dtw_execute() 呼叫完全成功（傳回零的值）
- dtw_execute() 常式設定 dtw_lei 結構中的 DTW_LE_CONTINUE 旗號。

當 dtw_getNextRow() 函數被呼叫時，dtw_lei 結構中的橫列欄位會被設定為指向橫列物件的指標。要處理橫列物件，您應該使用 Net.Data 公用程式常式 dtw_row_SetCols() 與 dtw_row_SetV()。它假設當第一次呼 dtw_getNextRow() 常式後，橫列物件會含有直欄標題。後續的呼叫應該含有實際的表格資料。

dtw_getNextRow() 繼續被呼叫（除非訊息區塊處理有另外指示），只要設定了 DTW_LE_CONTINUE 旗號的話。

常式 dtw_cleanup()

當處理 Net.Data 要求時，Net.Data 會呼叫一次語言環境的 dtw_cleanup() 函數(如果您有定義它的話)。函數會只在下列情況時被呼叫：如果 Net.Data 處理參照語言環境的 Net.Data 巨集而且 Net.Data 處理停止或有錯誤致使 Net.Data 停止處理 Net.Data 巨集。

如果清除處理發生異常的話,Net.Data 會將 dtw_lei 結構中的旗號欄位設定為 DTW_END_ABNORMAL，。以下是一些的異常狀況：

- 語言環境常式利用設定 dtw_lei 結構中的旗號欄位的 DTW_LE_FATAL_ERROR 位元來指示發生嚴重錯誤。
- Net.Data 發現無法回復的錯誤。
- Net.Data 巨集訊息區塊處理造成一個跳出。

如果語言環境的介面常式已經設定 le_opaque_data 欄位，則它必須使用 dtw_cleanup() 常式來釋放。

這個呼叫不會影響訊息區塊處理。如果傳回值不是零且有預設訊息要被發出，則該訊息會被發出；如果沒有預設訊息，則巨集處理器會發出一個警告訊息。

語言環境公用程式函數

Net.Data 公用程式函數可分成四個種類：

- 記憶體管理
- 架構變數管理
- 表格處理
- 橫列處理

記憶體管理函數用來配置 Net.Data 擁有的儲存體，以及釋放 Net.Data 使用 Net.Data 的執行程式庫配置的儲存體。下列範例示範這些介面的必要項目。假設 Net.Data 撰寫成使用編譯器 A 同時擁有對應的執行程式庫。程式設計師撰寫新的語言環境時卻使用編譯器 B 而且使用不同的執行程式庫。由於二個執行程式庫之間隱含的不相容，該語言環境無法釋放 Net.Data 配置的儲存體，Net.Data 也無法釋放該語言環境配置的儲存體。

表 2. 記憶體公用程式函數

第25頁的『dtw_malloc()』	使用 Net.Data 的執行程式庫從 Net.Data 的執行堆集配置儲存體。
第23頁的『dtw_free()』	使用 Net.Data 的執行程式庫（runtime library）來釋放從 Net.Data 的執行堆集配置的儲存體。
第29頁的『dtw_strdup()』	使用 Net.Data 的執行時間程式庫，從 Net.Data 的執行時間堆集中配置儲存體，並將所指定的字串複製到所配置的儲存體中。

架構變數之管理函數可以使語言環境存取 Net.Data 起始設定檔中儲存的架構資訊。這些函數可讓所有語言環境共用 Net.Data 起始設定檔，並使用其中的資訊來架構語言環境。

表 3. 架構公用程式函數

第24頁的『dtw_getvar()』	取回 Net.Data 起始設定檔中架構變數的值。
第28頁的『dtw_setvar()』	在 Net.Data 起始設定檔中，設定架構變數的值。

表格函數是用來操控任何已傳遞給語言環境的 Net.Data 巨集表格變數。

表 4. 表格公用程式函數

第40頁的 『dtw_table_New()』	建立一個表格物件
----------------------------	----------

表 4. 表格公用程式函數 (繼續)

第32頁的 『dtw_table_Delete()』	刪除一個表格物件。
第43頁的 『dtw_table_SetCols()』	設定表格的寬度，並配置直欄表頭的儲存體。
第36頁的 『dtw_table_GetV()』	取回表格橫列/直欄的值。
第45頁的 『dtw_table_SetV()』	設定表格橫列/直欄的值。
第35頁的 『dtw_table_GetN()』	取回一個表格直欄標題
第44頁的 『dtw_table_SetN()』	設定表格直欄標題。
第42頁的 『dtw_table_Rows()』	取得表格中現行橫列數目。
第31頁的 『dtw_table_Cols()』	取得表格中現行欄位數目。
第39頁的 『dtw_table_MaxRows()』	取得表格中允許橫列的最大數目
第41頁的 『dtw_table_QueryColNoNj()』	取得欄位標題的欄位號碼。
第30頁的 『dtw_table_AppendRow()』	在表格尾端新增一列或多列。
第38頁的 『dtw_table_InsertRow()』	在表格中插入一列或多列。
第34頁的 『dtw_table_DeleteRow()』	從表格中刪除一列或多列。
第37頁的 『dtw_table_InsertCol()』	在表格中插入一欄或多欄。
第33頁的 『dtw_table_DeleteCol()』	從表格中刪除一欄或多欄。

橫列函數可操控當循序處理橫列時，傳遞給語言環境的 dtw_getNextRow() 函數的橫列物件

表 5. 橫列公用程式函數

第26頁的 『dtw_row_SetCols()』	設定橫列的寬度。
第27頁的 『dtw_row_SetV()』	設定橫列/直欄值。

dtw_free()

使用

使用 `Net.Data` 的執行時間程式庫，釋放已從 `Net.Data` 執行時間堆集中配置的儲存體。緩衝區會指向將釋放的已配置儲存體。

格式

```
void dtw_free(void *buffer)
```

參數

buffer 指標指向將釋放的配置記憶體。

範例

```
char *myBuf;
long  nbytes = 8192;

myBuf = (char *)dtw_malloc(nbytes);

dtw_free((void *)myBuf);
```

dtw_getvar()

使用

從 Net.Data 起始設定檔中，取回 var_name 所指定的架構變數的值。

格式

```
char *dtw_getvar(char *var_name)
```

參數

var_name	將取回的架構變數的名稱。
----------	--------------

範例

```
char *myBindFile;

myBindFile = dtw_getvar("BIND_FILE");
```

dtw_malloc()

使用

這個函數將傳回一個指向使用 `Net.Data` 執行程式庫，從 `Net.Data` 執行堆集中配置的儲存體的指標。儲存體的長度為 `nbytes`。如果未傳回所要求的儲存體，將傳回 `NULL` 指標。

格式

```
void *dtw_malloc(long nbytes)
```

參數

nbytes 將配置的位元組數目。

範例

```
char *myBuf;
long  nbytes = 8192;

myBuf = (char *)dtw_malloc(nbytes);
```

dtw_row_SetCols()

使用

這個函數可設定橫列的寬度，並配置儲存體，直欄欄題使用。 dtw_row_SetCols() 函數可對每一列使用一次。

格式

```
int dtw_row_SetCols(void *row, int cols)
```

參數

row	指向尚未配置任何直欄的剛建立橫列的指標。
cols	將在新的橫列中配置的起始直欄數目。

範例

```
void *myRow;  
rc = dtw_row_SetCols(myRow, 5);
```


dtw_row_SetV()

使用

這個函數可設定欄位值。所指定的值應該是指向使用 `dtw_malloc()` 或 `dtw_strdup()` 所配置的字串的指標，或是將刪除欄位值的 `NULL`。

格式

```
int dtw_row_SetV(void *row, char *src, int col)
```

參數

<code>row</code>	將修改的橫列的指標。
<code>src</code>	含有將設定的新值的字串。
<code>col</code>	將設定的值的直欄號碼。

範例

```
void *myTable;  
char *myFieldValue = "newValue";  
  
rc = dtw_row_SetV(myRow, myFieldValue, 3);
```

dtw_setvar()

使用

這個函數可變更 Net.Data 起始設定檔中 `var_name` 所指定的架構變數的值。變數的新值將設定為 `new_value` 所指定的值。

格式

```
void dtw_setvar(char *var_name, char *new_value)
```

參數

<code>var_name</code>	將修改的架構變數的名稱。
<code>new_value</code>	設定 <code>var_name</code> 時將使用的新值。

範例

```
char *myVariableValue = "new variable value";  
dtw_setvar("MY_VARIABLE", myVariableValue);
```

dtw_strdup()

使用

這個函數可使用 `Net.Data` 的執行程式庫，從 `Net.Data` 的執行堆集中配置儲存體，並將所指定的字串複製到所配置的儲存體中。如果未傳回所要求的儲存體，將傳回 `NULL` 指標。

格式

```
char *dtw_strdup(char *string)
```

參數

`string` 將複製到已配置儲存體中的字串值的指標。

範例

```
char *myString = "This string will be duplicated.";
char *myDupString;

myDupString = dtw_strdup(myString);
```

dtw_table_AppendRow()

使用

這個函數可在表格尾端新增一列或多列。在新增後，必須使用 `dtw_table_SetV()` 來設定新列的欄位值。

格式

```
int dtw_table_AppendRow(void *table, int rows)
```

參數

table	指向將被附加橫列的表格的指標。
rows	將附加的橫列數目。

範例

```
void *myTable;  
rc = dtw_table_AppendRow(myTable, 10);
```

dtw_table_Cols()

使用

傳回表格中現行直欄數目。

格式

```
int dtw_table_Cols(void *table)
```

參數

table	指向將傳回其現行直欄數目的表格的指標。
-------	---------------------

範例

```
void *myTable;
int currentColumns;

currentColumns = dtw_table_Cols(myTable);
```

dtw_table_Delete()

使用

刪除所有欄位標題及欄位值，然後刪除表格物件。

格式

```
int dtw_table_Delete(void *table)
```

參數

table	將刪除的表格的指標。
-------	------------

範例

```
void *myTable;
```

```
rc = dtw_table_Delete(myTable);
```

dtw_table_DeleteCol()

使用

這個函數可刪除從 `start_col` 中所指定的欄位開始的一欄或多欄。欲刪除表格的所有橫列及欄位，請使用 `dtw_table_DeleteCol(table, 1, dtw_table_Cols());`。

格式

```
int dtw_table_DeleteCol(void *table, int start_col, int cols)
```

參數

<code>table</code>	指向將修改的表格的指標。
<code>start_col</code>	將刪除的第一個欄的直欄號碼。
<code>rows</code>	將刪除的列數。

範例

```
void *myTable;  
  
rc = dtw_table_DeleteCol(myTable, 1, 10);
```

dtw_table_DeleteRow()

使用

刪除從 start_row 中所指定的橫列開始的一列或多列。

格式

```
int dtw_table_DeleteRow(void *table, int start_row, int rows)
```

參數

table	將修改的表格的指標。
-------	------------

start_row	將刪除的第一列的橫列號碼。
-----------	---------------

rows	將刪除的橫列數目。
------	-----------

範例

```
void *myTable;
```

```
rc = dtw_table_DeleteRow(myTable, 3, 10);
```


dtw_table_GetN()

使用

取回直欄標題。經由透過 dtw_tableSetN 指定 NULL 給直欄標題，僅刪除所傳回的字串。

格式

```
int dtw_table_GetN(void *table, char **dest, int col)
```

參數

table	指向將從其中取回直欄標題的表格的指標。
dest	指向含有直欄標題的字串的指標。
col	直欄標題的直欄號碼。

範例

```
void *myTable;  
char *myColumnHeading;  
  
rc = dtw_table_GetN(myTable, &myColumnHeading, 5);
```

dtw_table_GetV()

使用

從表格中取回欄位值。您可以經由透過 dtw_tableSetV 指定 NULL 給欄位值，僅刪除所傳回的字串。

格式

```
int dtw_table_GetV(void *table, char **dest, int row, int col)
```

參數

table	指向將從其中取回一個值的表格的指標。
dest	指向含有值的字串的指標。
row	將取回的值的橫列號碼。
col	將取回的值的直欄號碼。

範例

```
void *myTable;  
char *myTableValue;  
  
rc = dtw_table_GetV(myTable, &myTableValue, 3, 5);
```

dtw_table_InsertCol()

使用

這個函數可在指定的直欄後，插入一欄或多欄。

格式

```
int dtw_table_InsertCol(void *table, int after_col, int cols)
```

參數

table	指向將修改的表格的指標。
after_col	將在其後插入新直欄的直欄號碼。欲在表格起始處插入欄位，請指定 0。
cols	將插入的直欄數目。

範例

```
void *myTable;  
  
rc = dtw_table_InsertCol(myTable, 3, 10);
```

dtw_table_InsertRow()

使用

在指定的橫列後，插入一系列或多列。

格式

```
int dtw_table_InsertRow(void *table, int after_row, int rows)
```

參數

table	指向將修改的表格的指標。
after_row	將在其後插入新橫列的橫列的號碼。 欲在表格起始處插入橫列，請對這個參數指定 0。
rows	將插入的橫列數目。

範例

```
void *myTable;  
  
rc = dtw_table_InsertRow(myTable, 3, 10);
```

dtw_table_MaxRows()

使用

傳回表格中所容許的最大橫列數目。

格式

```
int dtw_table_MaxRows(void *table)
```

參數

table 將傳回其最大橫列數目的表格的指標。

範例

```
void *myTable;  
int maximumRows;  
  
maximumRows = dtw_table_MaxRows(myTable);
```

dtw_table_New()

使用

這個函數可建立一個表格物件，並將所有直欄標題及欄位值起始設定為 `NULL`。呼叫者將指定橫列及直欄的起始數目，以及橫列的最大數目。如果橫列或欄位的起始數目為 0，則在呼叫任何表格函數之前，`dtw_table_SetCols()` 必須設定橫列中的欄位數目。

格式

```
int dtw_table_New(void **table, int rows, int cols, int row_lim)
```

參數

table	指向將建立的表格的指標。
rows	將在新表格中配置的起始橫列數目。
cols	將在新表格中配置的起始直欄數目。
row_lim	這個表格中可配置的橫列的最大數目。

範例

```
void *myTable;  
  
rc = dtw_table_New(&myTable, 20, 5, 100);
```

dtw_table_QueryColnoNj()

使用

傳回直欄標題之直欄號碼。

格式

```
int dtw_table_QueryColnoNj(void *table, char *name)
```

參數

table	指向將查詢的表格的指標。
name	指定將傳回其直欄號碼的直欄標題的字串。如果表格中沒有直欄標題，將傳回 0。

範例

```
void *myTable;  
int columnNumber;  
  
columnNumber = dtw_table_QueryColnoNj(myTable, "column 1");
```

dtw_table_Rows()

使用

傳回表格中現行橫列數目。

格式

```
int dtw_table_Rows(void *table)
```

參數

table	將傳回其現行橫列數目的表格的指標。
-------	-------------------

範例

```
void *myTable;
    int currentRows;

    currentRows = dtw_table_Rows(myTable);
```


dtw_table_SetCols()

使用

設定表格的直欄數目，並配置儲存體，供直欄標題使用。當建立表格時，或在使用其他表格函數之前，必須先呼叫這個函數來指定它時，指定直欄標題。dtw_table_SetCols() 函數僅可對表格使用一次。此後，使用 dtw_table_DeleteCol() 或 dtw_table_InsertCol() 函數。

格式

```
int dtw_table_SetCols(void *table, int cols)
```

參數

table	指向沒有配置任何直欄或橫列的新表格的指標。
cols	將在新表格中配置的起始直欄數目。

範例

```
void *myTable;  
  
rc = dtw_table_SetCols(myTable, 5);
```

dtw_table_SetN()

使用

設定直欄標題。指定一個指向將使用 `dtw_malloc()` 或 `dtw_strdup()` 來配置的字串的指標的值，或是指定 `NULL` 來刪除直欄標題。

格式

```
int dtw_table_SetN(void *table, char *src, int col)
```

參數

<code>table</code>	將設定其直欄標題的表格之指標。
<code>src</code>	含有將設定的新直欄標題之字串。
<code>col</code>	將設定之直欄標題的直欄號碼。

範例

```
void *myTable;  
char *myColumnHeading = "newColumnHeading";  
  
rc = dtw_table_SetN(myTable, myColumnHeading, 5);
```

dtw_table_SetV()

使用

設定欄位值。所指定的值是指向使用 `dtw_malloc()` 或 `dtw_strdup()` 所配置的字串的指標，或是將刪除欄位值的 `NULL`。

格式

```
int dtw_table_SetV(void *table, char *src, int row, int col)
```

參數

<code>table</code>	指向將設定其值的表格的指標。
<code>name</code>	含有新值的字串。
<code>row</code>	新值的橫列號碼。
<code>col</code>	新值的直欄號碼。

範例

```
void *myTable;  
char *myTableValue = "newValue";  
  
rc = dtw_table_SetV(myTable, myTableValue, 3, 5);
```


附錄. 語言環境模版

您可以使用這個模版，來建立自己的語言環境，以便善用 Net.Data。

```

/*****
/*
/* File Name
/*
/* Description
/*
/* Functions
/*
/* Entry Points
/*
/* Change Activity
/*
/*
/* Flag Reason Date Developer Description
/*
/*
/*
*****/

/*-----*/
/* Includes
/*-----*/
#include "dtwle.h"
/*-----*/
/* Function
/* dtw_getFp
/*
/* Purpose
/* Set function pointers to all Language Environment Interface
/* routines being provided by this Language Environment. If a
/* routine in the structure is not being provided, set that field
/* to NULL.
/*
/*
/* Format
/* int dtw_getFp(dtw_fp_t *func_pointer)
/*
/* Parameters
/* func_pointer A pointer to a structure which will contain
/* function pointers for all functions provided
/* by this language environment.
/*
/* Returns
/* Success ..... 0
/* Failure ..... -1
/*-----*/
int dtw_getFp(dtw_fp_t *func_pointer)
{
    func_pointer->dtw_initialize_fp = dtw_initialize;
    func_pointer->dtw_execute_fp = dtw_execute;
    func_pointer->dtw_getNextRow_fp = dtw_getNextRow;
    func_pointer->dtw_cleanup_fp = dtw_cleanup;
    return 0;
}
#endif
```

圖 1. 語言環境模版 (1/12)

```

/*-----*/
/*
/* Function
/*   dtw_initialize
/*
/* Purpose
/*
/* Format
/*   int dtw_initialize(dtw_lei_t *le_interface)
/*
/* Parameters
/*   le_interface    A pointer to a structure containing the
/*                   following fields:
/*
/*       function_name
/*       flags
/*       exec_statement
/*       parm_data_array
/*       default_error_message
/*       le_opaque_data
/*       row
/*
/* Returns
/*   Success ..... 0
/*   Failure ..... 0
/*-----*/
int dtw_initialize(dtw_lei_t *le_interface)
{
    return rc;
}

```

圖 1. 語言環境模版 (2/12)

```

/*-----*/
/*
/* Function
/*   dtw_execute
/*
/* Purpose
/*
/* Format
/*   int dtw_execute(dtw_lei_t *le_interface)
/*
/* Parameters
/*   le_interface      A pointer to a structure containing the
/*                     following fields:
/*
/*       function_name
/*       flags
/*       exec_statement
/*       parm_data_array
/*       default_error_message
/*       le_opaque_data
/*       row
/*
/* Returns
/*   Success ..... 0
/*   Failure ..... 0
/*-----*/
int dtw_execute(dtw_lei_t *le_interface)
{
    /*-----*/
    /* Determine if %exec statement was specified.
    /*-----*/
    if (le_interface->flags & DTW_STMT_EXEC) {
        /*-----*/
        /* Parse the %exec statement
        /*-----*/
        rc = processExecStmt(le_interface->exec_statement);
        if (rc)
        {
        }
    }
    else {
        /*-----*/
        /* Parse the inline data
        /*-----*/
        rc = processInlineData(le_interface->exec_statement);
        if (rc)
        {
        }
    }
}

```

圖 1. 語言環境模版 (3/12)

```

/*-----*/
/* Parse the input parameters */
/*-----*/
rc = processInputParms(le_interface->parm_data_array);
if (rc)
{
}
/*-----*/
/* Process the request */
/*-----*/
rc = processRequest();
if (rc)
{
}
/*-----*/
/* Process the output data */
/*-----*/
rc = processOutputParms(le_interface->parm_data_array);
if (rc)
{
}
/*-----*/
/* Process the return code and default error message */
/*-----*/
if (rc)
{
    setErrorMessage(rc, &(le_interface->default_error_message));
}
/*-----*/
/* Cleanup and exit program. */
/*-----*/
return rc;
}

```

圖 1. 語言環境模版 (4/12)


```

/*-----*/
/*
/* Function
/*   dtw_getNextRow
/*
/* Purpose
/*
/* Format
/*   int dtw_getNextRow(dtw_lei_t *le_interface)
/*
/* Parameters
/*   le_interface    A pointer to a structure containing the
/*                   following fields:
/*
/*       function_name
/*       flags
/*       exec_statement
/*       parm_data_array
/*       default_error_message
/*       le_opaque_data
/*       row
/*
/* Returns
/*   Success ..... 0
/*   Failure ..... 0
/*-----*/
int dtw_getNextRow(dtw_lei_t *le_interface)
{
    return rc;
}

```

圖 1. 語言環境模版 (5/12)

```

/*-----*/
/*
/* Function
/*   dtw_cleanup
/*
/* Purpose
/*
/* Format
/*   int dtw_cleanup(dtw_lei_t *le_interface)
/*
/* Parameters
/*   le_interface      A pointer to a structure containing the
/*                     following fields:
/*
/*       function_name
/*       flags
/*       exec_statement
/*       parm_data_array
/*       default_error_message
/*       le_opaque_data
/*       row
/*
/* Returns
/*   Success ..... 0
/*   Failure ..... 0
/*-----*/
int dtw_cleanup(dtw_lei_t *le_interface)
{
    /*-----*/
    /* Determine if this is normal or abnormal termination.
    /*-----*/
    if (le_interface->flags & DTW_END_ABNORMAL) {
        /*-----*/
        /* Do abnormal termination cleanup.
        /*-----*/
    }
    else {
        /*-----*/
        /* Do normal termination cleanup.
        /*-----*/
    }

    return rc;
}

```

圖 1. 語言環境模版 (6/12)

```

/*-----*/
/*
/* Function
/* processInputParms
/*
/* Purpose
/*
/*
/* Format
/* unsigned long processInputParms(dtw_parm_data_t *parm_data)
/*
/* Parameters
/* dtw_parm_data_t *parm_data
/*
/* Returns
/* Success ..... 0
/* Failure .....
/*
/*-----*/
unsigned long processInputParms(dtw_parm_data_t *parm_data)
{
    /*-----*/
    /* Loop through all the variables in the parameter data array. */
    /* The array is terminated by a NULL entry, meaning the parm_name */
    /* field is set to NULL, the parm_value field is set to NULL, and */
    /* the parm_descriptor field is set to 0. However, the only valid */
    /* check for the end of the parameter data array is to check */
    /* parm_descriptor == 0, since the parm_name field is NULL when a */
    /* literal string is passed in, and the parm_value field is set */
    /* to NULL when an undeclared variable is passed in. */
    /*-----*/
    for (; parm_data->parm_descriptor != 0; ++parm_data) {

```

圖 1. 語言環境模版 (7/12)

```

/*-----*/
/* Determine the usage of each input parameter.      */
/*-----*/
switch(parm_data->parm_descriptor & DTW_USAGE) {

    case(DTW_IN):
        /*-----*/
        /* Determine the type of each input parameter.    */
        /*-----*/
        switch (parm_data->parm_descriptor & DTW_TYPE) {
            case DTW_STRING:
                break;
            case DTW_TABLE:
                break;
            default:
                /*-----*/
                /* Internal error - unknown data type      */
                /*-----*/
                break;
        }
        break;

    case(DTW_OUT):
        break;

    case(DTW_INOUT):
        break;

    default:
        /*-----*/
        /* Internal error - unknown usage                  */
        /*-----*/
        break;
}
}
return rc;
}

```

圖 1. 語言環境模版 (8/12)

```

/*-----*/
/*
/* Function
/*   processOutputParms()
/*
/* Purpose
/*
/* Format
/*   unsigned long processOutputParms(dtw_parm_data_t *parm_data)
/*
/* Parameters
/*   dtw_parm_data_t *parm_data
/*
/* Returns
/*   Success ..... 0
/*   Failure ..... -1
/*
/*-----*/
unsigned long processOutputParms(dtw_parm_data_t *parm_data) {
    /*-----*/
    /* Get output data in some language environment-specific manner. */
    /* This is entirely dependent on what the language environment */
    /* is interfacing to, and how the LE chooses to interface to it. */
    /*-----*/

    /*-----*/
    /* Loop through all the parms in the parameter data array,
    /* looking for output parameters.
    /*-----*/
    for (; parm_data->parm_descriptor != 0; ++parm_data) {

        /*-----*/
        /* Determine usage of each parameter.
        /*-----*/
        if (pd_i->parm_descriptor & DTW_OUT) {
            /*-----*/
            /* Determine the type of each input parameter.
            /*-----*/
            switch (pd_i->parm_descriptor & DTW_TYPE) {
                case DTW_STRING:
                    /*-----*/
                    /* Give a string parameter a new value. If the
                    /* parameter value is not currently NULL, the
                    /* storage must be freed using an LE interface
                    /* utility function if it was allocated by
                    /* Net.Data.
                    /*-----*/
                    if (parm_data->parm_value != NULL)
                        dtw_free(parm_data->parm_value);
                    parm_data->parm_value = dtw_strdup(newValue);
                    break;

```

圖 1. 語言環境模版 (9/12)

```

case DTW_TABLE:
/*-----*/
/* Change the size of a table parameter. Use the */
/* LE interface utility functions to modify the */
/* table object. */
/*-----*/
/*-----*/
/* First get the pointer to the table object. */
/*-----*/
void *myTable = (void *) parm_data->parm_value;
/*-----*/
/* Next get the current size of the table. */
/*-----*/
cols = dtw_table_Cols(myTable);
rows = dtw_table_Rows(myTable);
/*-----*/
/* Now set the new size (assumes the new size */
/* values are valid). */
/*-----*/

/*-----*/
/* Set the columns first. */
/*-----*/
if (cols > newColValue)
{
    dtw_table_DeleteCol(myTable,
                        newColValue + 1,
                        cols - newColValue);
}
else if (cols < new_col_value)
{
    dtw_table_InsertCol(myTable,
                        cols,
                        newColValue - cols);
}

/*-----*/
/* Now set the rows. */
/*-----*/
if (newColValue > 0) {
    if (rows > newRowValue)
    {
        dtw_table_DeleteRow(myTable,
                            newRowValue + 1,
                            rows - newRowValue);
    }
    else if (rows < new_row_value)
    {
        dtw_table_InsertRow(myTable,
                            rows,
                            newRowValue - rows);
    }
}
}

```

圖 1. 語言環境模版 (10/12)

```

/*-----*/
/* Now get the last row/column value.          */
/*-----*/
dtw_table_GetV(myTable,
               &myValue;,
               newRowValue,
               newColValue);

/*-----*/
/* Delete the last row/column value.          */
/*-----*/
dtw_table_SetV(myTable,
               NULL,
               newRowValue,
               newColValue);

/*-----*/
/* Set the last row/column value.             */
/*-----*/
dtw_table_SetV(myTable,
               dtw_strdup(myNewValue),
               newRowValue,
               newColValue);

break;
default:
/*-----*/
/* Internal error - unknown data type          */
/*-----*/
break;
    }
}
}
return 0;
}

```

圖 1. 語言環境模版 (11/12)

```

/*-----*/
/*
/* Function
/*   setErrorMessage()
/*
/* Purpose
/*
/* Format
/*   unsigned long setErrorMessage(int returnCode,
/*                                   char **defaultErrorMessage)
/*
/* Parameters
/*   int   returnCode
/*   char **defaultErrorMessage
/*
/* Returns
/*   Success ..... 0
/*   Failure ..... -1
/*
/*-----*/
unsigned long setErrorMessage(int returnCode,
                             char **defaultErrorMessage)
{
    /*-----*/
    /* Set the default error message based on the return code.
    /*-----*/
    switch(returnCode) {
        case LE_SUCCESS:
            break;
        case LE_RC1:
            *defaultErrorMessage = dtw_strdup(LE_RC1_MESSAGE_TEXT);
            break;
        case LE_RC2:
            *defaultErrorMessage = dtw_strdup(LE_RC2_MESSAGE_TEXT);
            break;
        case LE_RC3:
            *defaultErrorMessage = dtw_strdup(LE_RC3_MESSAGE_TEXT);
            break;
        case LE_RC4:
            *defaultErrorMessage = dtw_strdup(LE_RC4_MESSAGE_TEXT);
            rc = LE_RC1INTERNAL;
            break;
    }
    return 0;
}

```

圖 1. 語言環境模版 (12/12)

索引

索引順序以中文字，英文字，及特殊符號之次序排列。

〔八劃〕

函數

- dtw_free() 23
- dtw_getvar() 24
- dtw_malloc() 25
- dtw_row_SetCols() 26
- dtw_row_SetV() 27
- dtw_setvar() 28
- dtw_strdup() 29
- dtw_table_AppendRow() 30
- dtw_table_Cols() 31
- dtw_table_DeleteCol() 33
- dtw_table_DeleteRow() 34
- dtw_table_Delete() 32
- dtw_table_GetN() 35
- dtw_table_GetV() 36
- dtw_table_InsertCol() 37
- dtw_table_InsertRow() 38
- dtw_table_MaxRows() 39
- dtw_table_New() 40
- dtw_table_QueryColnoNj() 41
- dtw_table_Rows() 42
- dtw_table_SetCols() 43
- dtw_table_SetN() 44
- dtw_table_SetV() 45

呼叫 applet 4

〔九劃〕

架構環境 12

〔十一劃〕

常式

- dtw_cleanup() 20
- dtw_execute() 20
- dtw_getNextRow() 20
- dtw_initialize() 19

〔十二劃〕

結構

- dtw_lei 17

結構 (繼續)

- dtw_parm_data 18

〔十四劃〕

語言環境

- 系統 11
- 建立 15
- 架構 12
- 結構 17
- 摘要 3
- Java applet 4
- Java 應用程式 8
- ODBC 9
- Oracle 9
- PERL 10
- REXX 10
- SQL 11
- Sybase 11

〔十五劃〕

模版 47

〔十六劃〕

錯誤狀況 16

D

- DTW_JAVAPPS 8
- DTW_ODBC 9
- DTW_ORA 9
- DTW_PERL 10
- DTW_REXX 10
- DTW_SQL 11
- DTW_SYB 11
- DTW_SYSTEM 11

J

- Java applet 4
- 呼叫 4
- 參數 5
- 類別 7