

SBSH PocketWeather Custom Weather Feed Manual

For PocketWeather 2.3.0



By: Jenny Oliver

Last Updated: 16 June 2009

Downloads and additional support can be found at our site

<http://www.sbsh.net>

Table of Contents

Overview.....	3
Files and Folders.....	3
File Content.....	3
Custom Weather Feed Language Tutorial.....	4
main Function.....	6
Obtaining the Html Data From the URL Address.....	7
Extracting data from the Html Text.....	9
Selecting an Area of the HTML Data.....	10
Getting the Weather Forecast for Each Day.....	11
While Control Statement:	12
If Control Statement:	12
Adding the Obtained Variables into PocketWeather.....	12
Additional Examples	13
Map Variables.....	13
Checking for Today's Weather.....	13
User Reference Guide.....	14
Functions.....	14
Data Types.....	15
Variable Scope.....	16
Expressions.....	17
Statements.....	18
Comments.....	18
Stages of Execution of a CWF.....	19
Built-In Functions.....	19
Sky Conditions.....	28
SkyTextID Conditions Text.....	29
CWF Development Tool.....	31
Compilation.....	31
Debugging.....	32

Overview

This manual describes the definition of PocketWeather custom weather feed (CWF) files. A CWF can be created in order to obtain weather data from a specific web-site.

CWFs are defined using a simple structured language.

A tutorial example illustrates the usage of the CWF language. A user-reference guide explains the details of each command in the CWF language. The final section introduces the PocketWeather CWF development PC tool.

Files and Folders

PocketWeather custom weather feed files, must end with the ".CWF" file extension. These are simple plain-text files which can be edited with your favourite text editor, such as Notepad. Although ANSI text encoding is allowed, it is strongly recommended that UNICODE (big endian) encoding be used.

To install a new CWF file onto your device, simply copy the ".CWF" file into the "Templates" folder on your device.

To make organisation of the files easier, it is recommended that new folders (or sub-folders) be added under the "Templates" folder. For example, you could add a folder called "WeatherCom" under "Templates".

File Content

The CWF file contains a series of instructions or commands that tell PocketWeather how to extract data from a given weather feed.

Custom Weather Feed Language Tutorial

The following is a tutorial walk-through of an example CWF definition. The CWF definition is for handling forecasts provided by **weather.com** :

```
// CWF for 10 day forecast from weather.com
main
{
    pointer weatherTenDayStart
    {
        go_start;
        skip_to_text("new mpdFObj", true);
        skip_to_text(";", true);
    };

    pointer weatherTenDayEnd
    {
        skip_to_text("graphTrgs", true);
    };

    string DELIMITER = "'", ";

    // downloading and parsing 10 days forecast
    {
        open_url("http://www.weather.com/weather/mpdwcr/tenday?locid=" + CityCode +
            "&channel=dailytraveler&datapoint=htempdp&adprodname=lap_travel_daily_main",
            "GET");

        skip_to_text("new mpdHeadObj(", true);
        skip_chars(DELIMITER);

        // temperature units
        begin_var;
        skip_to_chars("'");
        string tmpUnits = end_var;

        // wind speed units
        skip_to_text(",", true); skip_to_text(",", true); skip_chars(DELIMITER);
        begin_var;
        skip_to_chars("'");
        string windSpeedUnits = end_var;

        select_area(weatherTenDayStart, weatherTenDayEnd);

        string sky = ".";

        while (sky)
        {
            sky = "";
            // day of month
            skip_to_text("new Date(", true);
            skip_to_text(",", true); skip_to_text(",", true); skip_chars(DELIMITER);
            begin_var;
            skip_to_chars("'");
            string dayOfMonth = end_var;

            // temperature high
            skip_to_text(")", true);
            skip_to_text(",", true); skip_to_text(",", true); skip_chars(DELIMITER);
            begin_var;
            skip_to_chars("'");
            string tmpHi = end_var;

            // temperature low
```

```

skip_to_text(",", true); skip_chars (DELIMITER);
begin_var;
skip_to_chars("'");
string tmpPrLo = end_var;

// sky
skip_to_text(",", true); skip_to_text(",", true); skip_chars (DELIMITER);
begin_var;
skip_to_chars("'");
sky = end_var;

// wind speed
skip_to_text(",", true); skip_to_text(",", true); skip_chars (DELIMITER);
begin_var;
skip_to_chars("'");
string windSpeed = end_var;

// wind direction
skip_to_text(",", true); skip_to_text(",", true); skip_chars (DELIMITER);
begin_var;
skip_to_chars("'");
string windDirection = end_var;

// precipitation
skip_to_text(",", true); skip_chars (DELIMITER);
begin_var;
skip_to_chars("'");
string precipitation = end_var;

// humidity
skip_to_text(",", true); skip_chars (DELIMITER);
begin_var;
skip_to_chars("'");
string humidity = end_var;

if (sky)
{
    add_day;
}
}
}
}

```

main Function

The entire content of the CWF definition is inserted between the brackets: **{}** of the **main** function including all variables and other functions.

```
main
{
}
```

Obtaining the Html Data From the URL Address

In order to download weather information from weather.com we need to provide the correct URL address e.g.:

http://www.weather.com/weather/mpdwcr/tenday?locid=CityCode&channel=dailytraveler&datapoint=hempdp&adprodname=lap_travel_daily_main

Where **CityCode** is the code used to identify a specific location. The ICAO code for Littlehampton, UK is **UKXX0260**:

http://www.weather.com/weather/mpdwcr/tenday?locid=UKXX0260&channel=dailytraveler&datapoint=hempdp&adprodname=lap_travel_daily_main

If you follow this link you will get a weather forecast for Littlehampton from weather.com.

In this example, a correct address consists of three parts: a common part at the beginning and at the end for all addresses, with a city code in between. We can insert the following line:

```
main
{
    ...
    open_url("http://www.weather.com/weather/mpdwcr/tenday?locid=" + CityCode +
        "&channel=dailytraveler&datapoint=hempdp&adprodname=lap_travel_daily_main",
        "GET");
    ...
}
```

The URL address is specified as an argument of the **open_url** function. The first argument of this function is a string consisting of the initial common part of the address line, the **CityCode**, and a further common part. **CityCode** is a predefined variable in PocketWeather. The strings are concatenated with a "+" operator. In some cases the address line can be more complicated. All functions, variables, and operators that you will use for string handling are described in this manual.

The following is the HTML data found at the above URL address for Littlehampton:

```
mpdData['head']=new mpdHeadObj('F', 'mi', 'mph', 'in', 'in');
mpdData['loc']=new mpdLocObj('UKXX0260', '1', '', 'Littlehampton, United Kingdom',
    new Date('2008', '4', '26', '4', '19', '48'), 'Local Time', '', 'ENG', '', 'UK', 'United
    Kingdom', 'England', '', 'Littlehampton');
mpdData['dayf']=new mpdFObj('May 26, 4:19 AM Local Time', false);
mpdData['dayf'].day[0]=new mpdFDObj(new Date('2008', '4', '26', '0', '0', '0'),
    'Today', '58', '52', '4', '40', 'Heavy Rain', '18', 'East Northeast', 'ENE', '80', '85');
mpdData['dayf'].day[1]=new mpdFDObj(new Date('2008', '4', '27', '0', '0', '0'),
    'Tue', '68', '53', '4', '11', 'Light Rain / Fog', '9', 'Northeast', 'NE', '60', '78');
mpdData['dayf'].day[2]=new mpdFDObj(new Date('2008', '4', '28', '0', '0', '0'),
    'Wed', '66', '54', '5', '12', 'Rain', '16', 'Southeast', 'SE', '60', '83');
mpdData['dayf'].day[3]=new mpdFDObj(new Date('2008', '4', '29', '0', '0', '0'),
    'Thu', '66', '53', '6', '11', 'Few Showers', '10', 'South Southwest', 'SSW', '30', '75');
mpdData['dayf'].day[4]=new mpdFDObj(new Date('2008', '4', '30', '0', '0', '0'),
    'Fri', '67', '53', '6', '30', 'Partly Cloudy', '8', 'West Southwest', 'WSW', '20', '76');
mpdData['dayf'].day[5]=new mpdFDObj(new Date('2008', '4', '31', '0', '0', '0'),
    'Sat', '66', '52', '6', '30', 'Partly Cloudy', '8', 'Northwest', 'NW', '20', '77');
mpdData['dayf'].day[6]=new mpdFDObj(new Date('2008', '5', '1', '0', '0', '0'),
    'Sun', '66', '52', '7', '30', 'Partly Cloudy', '8', 'Northwest', 'NW', '0', '71');
mpdData['dayf'].day[7]=new mpdFDObj(new Date('2008', '5', '2', '0', '0', '0'),
    'Mon', '66', '52', '6', '30', 'Partly Cloudy', '9', 'West', 'W', '10', '73');
mpdData['dayf'].day[8]=new mpdFDObj(new Date('2008', '5', '3', '0', '0', '0'),
    'Tue', '66', '52', '6', '30', 'Partly Cloudy', '9', 'East Southeast', 'ESE', '10', '71');
mpdData['dayf'].day[9]=new mpdFDObj(new Date('2008', '5', '4', '0', '0', '0'),
    'Wed', 'N/A', 'N/A', 'N/A', '-', 'N/A', 'N/A', 'N/A', 'N/A', 'N/A', 'N/A');
mpdErr=true;
mpdErrArray[mpdErrArray.length+1]='NO OR INVALID TRIGGER DATA IN REQUEST!';
graphTrgs=[];
mpdErr=true;
mpdErrArray[mpdErrArray.length+1]='NO OR INVALID DATA IN REQUEST!';
mpdData['dayfgraph']=new mpdTGOBJ();
svrWxAlertMode=0;
```

```
svrWxAlertType='';  
mpdData['nswxalrt']=new Array();  
mpdData['locvideo']=new  
mpdLocalVideoObj('false','null','/multimedia/videoplayer.html?  
collection=topstory&clip=365','Video');  
mpdErr=true;  
mpdErrArray[mpdErrArray.length+1]= 'INVALID ADLIST OBJECT!';
```


Extracting data from the Html Text

Before extracting the specific data for each day, first get some general data e.g. Temperature Units and the Wind Speed Units:

The text at the start of the HTML data is:

```
mpdData['head']=new mpdHeadObj('F', 'mi', 'mph', 'in', 'in');
```

Declare a string variable called **DELIMITER** with the value `"', "`:

```
string DELIMITER = "', ";
```

Then skip to the text `"new mpdHeadObj(" :`

```
skip_to_text("new mpdHeadObj(", true);
```

and skip over the characters defined in the string **DELIMITER** :

```
skip_chars(DELIMITER);
```

This moves the cursor to the data: `"F', 'mi', 'mph', 'in', 'in');"'`

We are now ready to extract the data for the Temperature units.

```
// temperature units
begin_var;
skip_to_chars("'");
string tmpUnits = end_var; // extracts tmpUnits="F"
```

The **begin_var** function saves the current cursor position.

The **skip_to_chars** function moves the cursor to the next `"' "`.

The string variable **tmpUnits** is declared and is assigned a value using the **end_var** function.

The **end_var** function extracts the text between the cursor position saved by the **begin_var** function up to the new current cursor position. Here, **tmpUnits** is assigned the value `"F"`.

This moves the cursor to the data: `"', 'mi', 'mph', 'in', 'in');"'`

Similarly, the **windSpeedUnits** are assigned to `"mph"` by skipping to `" , "` twice, skipping over the characters in the **DELIMITER** string and extracting the text up to the next `"' "`.

```
// wind speed units
skip_to_text(", ", true); skip_to_text(", ", true); skip_chars(DELIMITER);
begin_var;
skip_to_chars("'");
string windSpeedUnits = end_var; // extracts windSpeedUnits="mph"
```

Selecting an Area of the HTML Data

Once you have located the required URL address you are ready to examine the HTML data in more detail. The HTML data may be very complex. We need to find the specific data that we are interested in. First of all it may be helpful to select an important area of the data to analyse.

Let's scan the obtained html file for Littlehampton. Open the Littlehampton forecast page following the previously discussed link:

(http://www.weather.com/weather/mpdwcr/tenday?locid=UKXX0260&channel=dailytraveler&datapoint=htempdp&adprodname=lap_travel_daily_main) and ask your browser to show the document source. You will see the following html data:

We find that the weather data is contained between the text:

```
new mpdFObj (
```

and the next instance of the line:

```
graphTrgs
```

Now we can select the important area of data between these texts. This area can be selected by means of **select_area** function. This function has two arguments of **pointer** type: the beginning and the end of the data. Declare two new variables in our template and insert the **select_area** function as follows:

```
main
{
    pointer weatherTenDayStart
    {
        go_start;
        skip_to_text("new mpdFObj", true);
        skip_to_text(")", true);
    };

    pointer weatherTenDayEnd
    {
        skip_to_text("graphTrgs", true);
    };

    string DELIMITER = "'", " ";

    // downloading and parsing 10 days forecast
    {
        open_url("http://www.weather.com/weather/mpdwcr/tenday?locid=" + CityCode +
            "&channel=dailytraveler&datapoint=htempdp&adprodname=lap_travel_daily_main",
            "GET");
        ...
        select_area(weatherTenDayStart, weatherTenDayEnd);
    }
}
```

weatherTenDayStart: this variable of *pointer* type will put the cursor to the beginning of the important area of the document.

weatherTenDayEnd: this variable of *pointer* type will put the cursor to the end of the important area of the document.

go_start: this is a built-in function that puts the cursor at the beginning of a selected area, or, if no area is yet selected, or a new area is being selected, puts the cursor at the very beginning of the document

skip_to_text this function looks for given text from the current cursor position. If the text is found then the cursor moves to the end of the found fragment, otherwise the cursor doesn't move.

select_area: this function selects the area of text in the document to parse. All subsequent statements will operate within the selected area and never leave it. In this case this function selects the area between the two pointers: **weatherTenDayStart** and **weatherTenDayEnd**, as set above.

These and other functions, variables, operators are described in detail in the reference section of this manual.

Getting the Weather Forecast for Each Day

Now let's focus on the selected area and try to parse the forecast data from it. We can note that each day is represented with a repeatable block. Here is the data for the first day:

```
new Date('2008','4','26','0','0','0'),'Today','58','52','4','40','Heavy Rain','18','East  
Northeast','ENE','80','85');
```

The `skip_to_text`, `skip_chars`, `skip_to_chars`, `begin_var` and `end_var` functions are used as described earlier to extract the data into string variables. These variables are required to provide the data for the `add_day` function that will export the data to PocketWeather.

Follow the code through and see how the variables are assigned to the values shown in comments:

```
// day of month
skip_to_text("new Date(", true);
skip_to_text(",", true); skip_to_text(",", true); skip_chars(DELIMITER);
begin_var;
skip_to_chars("'");
string dayOfMonth = end_var;           // extracts dayOfMonth="26"

// temperature high
skip_to_text(")", true);
skip_to_text(",", true); skip_to_text(",", true); skip_chars(DELIMITER);
begin_var;
skip_to_chars("'");
string tmprHi = end_var;               // extracts tmprHi="58"

// temperature low
skip_to_text(",", true); skip_chars(DELIMITER);
begin_var;
skip_to_chars("'");
string tmprLo = end_var;               // extracts tmprLo="52"

// sky
skip_to_text(",", true); skip_to_text(",", true); skip_chars(DELIMITER);
begin_var;
skip_to_chars("'");
sky = end_var;                         // extracts sky="Heavy Rain"

// wind speed
skip_to_text(",", true); skip_to_text(",", true); skip_chars(DELIMITER);
begin_var;
skip_to_chars("'");
string windSpeed = end_var;            // extracts windSpeed="18"

// wind direction
skip_to_text(",", true); skip_to_text(",", true); skip_chars(DELIMITER);
begin_var;
skip_to_chars("'");
string windDirection = end_var;        // extracts windDirection="ENE"

// precipitation
skip_to_text(",", true); skip_chars(DELIMITER);
begin_var;
skip_to_chars("'");
string precipitation = end_var;         // extracts precipitation="80"

// humidity
skip_to_text(",", true); skip_chars(DELIMITER);
begin_var;
skip_to_chars("'");
string humidity = end_var;             // extracts humidity="85"
```

While Control Statement:

The **while** control statement is used to conditionally execute a single statement or compound statement (enclosed in **{}**). In the following code, the statement contained between **{}** will be executed repeatedly until the string variable **sky** is empty.

When the string variable **sky** becomes empty, control will pass to the code following the **while** statement.

```
string sky = ".";

while (sky)
{
    sky = "";
    ...
}
```

If Control Statement:

The **if** control statement is used to conditionally execute a single statement or compound statement (enclosed in **{}**). In the following code, the statement contained between **{}** will only be executed if the string variable **sky** is non-empty.

```
if (sky)
{
    add_day;
}
```

Adding the Obtained Variables into PocketWeather

Now we have five weather variables with new values. We now need to transfer this data into the PocketWeather program.

This is done as follows:

```
add_day;
```

This function doesn't take any parameters but assumes that the variables like **tmprHi**, **tmprLo** and **sky** have the correct values. The complete list of the variables used in **add_day** function will be described below.

Now we can do the same for each day by repeating the while loop:

Additional Examples

The following additional examples were not covered in the tutorial:

Map Variables

The following code shows an example definition of a map variable **skyMap**. The **key_value** function is used to associate a string value (2nd arg) with a given string key value (1st arg). Thus the string value "13" is associated with the string key value "1".

```
map skyMap
{
    key_value("1", "13");
    key_value("2", "13");
    ...
    key_value("47", "28");
    key_value("48", "14");
};
```

Later in the code we see the following statement:

```
sky = skyMap[sky];
```

The string variable **sky** is set up earlier. The value is now converted using the **skyMap** definition.

Checking for Today's Weather

There's one problem that should be mentioned. Some weather sources may provide expired data for yesterday. In order to skip such forecasts we can use this additional code:

```
string foundToday = ""; // false
string dayOfWeek = "";
begin_var;
skip_to_chars(".");
dayOfWeek = end_var;

if (!foundToday)
{
    if (is_today(dayOfWeek))
    {
        foundToday = "."; // true
    }
}
```

Declare a string variable **foundToday**, initialized to empty. Once the program finds a valid forecast for today, this variable gets the non-empty value, which also means "true"

We check if **foundToday** is not empty (an empty string in the template language returns "false" itself as well) and if it is we also check if the obtained value of **dayOfWeek** matches the current day (this is performed by means of the **is_today** function). If it doesn't match, then **foundToday** will remain empty and the **dayOfWeek** value will be incorrect. In this case the obtained day information will not be added, that's why we should add the following code in the end of the *while*-section:

```
if (foundToday)
{
    if (sky)
        add_day;
}
```


Data Types

There are 4 data types: `bool` (constants only), `string`, `map` and `pointer`.

Bool

`bool` constants (`true/false`) are passed as parameters to the `skip_to_text` and `skip_to_text_between_tags` functions.

String

`string` constants are contained in double quotes: `"My String"`. `String` variables are declared as follows:

```
string strDay = "Monday";      // string declared with initial value
string strText;                // string declared with no initial value
```

If a `string` constant contains the double-quote (") character, it must be preceded with the backslash symbol in the code: `"My name is \"Peter\""`. If the string contains the backslash ("\") character, it should be doubled: `"c:\\users\\johnsmith"`.

Map

`map` variables are defined as a number of key-value pairs each of which defines the mapping a string key to a new value. For example, to map `"sunny"` to `"1"`, `"rainy"` to `"2"` and `"cloudy"` to `"3"` you define a map variable as follows:

```
map mySkyMap
{
    key_value("sunny", "1");
    key_value("rainy", "2");
    key_value("cloudy", "3");
};
```

After this declaration the following code is valid:

```
string strSky = "rainy";
string strSkyCode = mySkyMap[strSky];
```

The `strSkyCode` variable will get the value `"2"`.

Pointer

`pointer` variables exists to mark an area of html data. This will be described with the `select_area` function.

Variable Scope

The scope of a variable declaration is local to the operational block in which the variable is defined. Example:

```
string str = "alpha";

if (str)
{
    string strOne = "one";
    if (strOne)
    {
        string strTwo = str + " - " + strOne;    // VALID
    }
}
string strThree = str + strTwo; // INVALID: strTwo is out of scope
```


Expressions

An expression in the CWF language uses a combination of **string** values and operators to yield a **string** result. The following elements are supported:

- **string** constants (e.g. "sunny")
- **string** variables (e.g. **strSky**)
- **map** variables (e.g. **mySkyMap[**strSky**]**)
- assignment operator "=" (e.g. **strSky** = "Sunny")
- concatenation operator "+"
- negation operator "!"
- equality operator "~"
- inequality operator "\$"
- round brackets "(" and ")";

The precedence of operators is as follows (from highest to lowest):

priority 4: []

priority 3: !

priority 2: +

priority 1: ~ \$

The concatenation "+" operator takes 2 **string** arguments and returns the **string** concatenation of the 2nd argument to the first. Example:

```
string strA = "123";  
string strB = "321";  
string strC = strA + strB;
```

The **strC** variable is assigned the value "123321".

The negation "!" operator takes one **string** argument. If the argument is an empty string, it returns a non-empty **string** result ("."). If the argument is a non-empty string, it returns an empty string result. Examples:

```
string str1 = "one";  
string str2 = !str1;
```

Then **str2** variable is assigned the empty **string** value "".

```
string str1 = "";  
string str2 = !str1;
```

Then **str2** variable is assigned the non-empty **string** value ".".

The equality "~" operator compares strings. It takes 2 **string** arguments. If the arguments are equal strings then it returns a nonempty string result ("."). If the arguments are not equal then it returns an empty **string** result.

The inequality operator "\$" works like the equality "~" operator but returns the opposite result.

Round brackets allow you to override the operator precedence. Example:

```
string strName = "my name";
string strOne = "111";
string strTwo = "two";
map myNumberMap
{
    key_value("one", "111");
    key_value("two", "222");
};
string str = strName + ("333" $ !(strOne + myNumberMap[strTwo]));
```

The order of execution is as follows:

```
myNumberMap[strTwo] will return "222",
strOne + myNumberMap[strTwo] will return "111222",
!(strOne + myNumberMap[strTwo]) will return "" (empty string),
"333" $ "" will return "." ("333" is not equal to ""),
strName + "." is "my name.". The value of the whole expression is "my name.".
```

Statements

There are two control statements in the CWF language: **if** (without **else**) and **while**. They take the value of string expression to control the execution of the following single statement or compound statement (enclosed by {}). A non-empty string expression value is considered true. An empty string value is considered false. Examples:

```
string str = "one";

if (str)
{
    // this code executes
}

string a = "12";
if (a + "3" ~ "123")
    a = "yes"; // this statement executes

string strCounter = "";
while (strCounter $ "aaaaa")
{
    // this code will be executed 5 times

    strCounter = strCounter + "a";
}
```

Comments

Comments can be included preceding the comment text by two "//" characters. The example code above includes comments.

Stages of Execution of a CWF

Specify Location

In order to extract weather data for a specific location, the predefined **CityCode** global variable must be defined. The **CityCode** variable will contain the appropriate code (existing in the PocketWeather database) to identify this location in the chosen weather source. The maximum length of the custom city code is 30 characters, case is any.

Load Weather Data to Parse

When the **main** function starts its execution, it has no data to parse. The data needs to be download from the web site via **open_url** function. It takes two arguments: the URL address and method (**"GET"** or **"POST"**). See the detailed description below.

How to Navigate Through the Data

Use the following functions: **go_start**, **select_area**, **skip_to_text_between_tags**, **skip_tags**, **skip_to_tags**, **skip_to_text**, **skip_chars**, **skip_to_chars** (see below).

How to Extract Data

Use **begin_var/end_var** functions (see below).

How to Transfer Data to the PocketWeather Database

Call the **add_day** function (see below).

Built-In Functions

In the CWF language, those functions that have no arguments don't require any brackets (moreover, brackets are not allowed).

```
void debug_log;
```

This function turns on debug logging to the file "WeatherParser_log.txt" that will be placed in the root folder of the Pocket PC. This is discussed in more detail in the debugging section at the end of this manual.

open_url

```
void open_url(string strURL, string strMethod);
```

This function downloads the data from the site given by URL. It takes two arguments: URL (strURL) and method (strMethod). Two methods are supported: "GET" and "POST". If the method is "GET" the form parameters are placed in the URL in a standard way. Example:

```
open_url("http://www.my.weather-source.com/cgi-bin/forecast.cgi?CityID=" + CityCode  
+ "&Forecast=10days", "GET");
```

If the method is "POST" the **open_url** function simulates a form submit. The form parameters should be filled separately by several calls to **form_param** function (see the description below). The form parameters list exists until the call to **open_url** function which uses and then clears it. Example:

```
form_param("CityID", CityCode);  
form_param("Forecast", "10days");  
open_url("http://www.my.weather-source.com/cgi-bin/forecast.cgi", "POST");
```

The **open_url** function supports up to 5 sequential redirects. If the received HTML contains the redirection to another URL then PocketWeather automatically opens this URL, and this operation is repeated up to 5 times. If the redirections still persist, **open_url** fails.

If the **open_url** fails (it cannot download data from site) then the CWF execution stops with error.

If the **open_url** succeeds, the internal document is filled and the cursor starts at the beginning of the document. After that the programmer can use the built-in functions to move the cursor forward or backward.

form_param

```
void form_param(string strName, string strValue);
```

This function defines the form parameter that will be passed to the server in the next **open_url** function call (useful if the method is "POST"). The function takes two arguments: strName and strValue – the name and the string value of the form parameter. See the example above.

begin_var and end_var

```
void begin_var;  
string end_var;
```

These functions obtain the data from the document.

Step 1. Move to the beginning of the data piece.

Step 2. Call **begin_var** function. This will remember the starting position in the document.

Step 3. Move forward through the data piece.

Step 4. Call **end_var** function. The function returns the trimmed string (without spaces at the start and at the end) from the document starting with the position remembered in the step 2 and finished in the current position.

Example. The document contains the date; we want to get the day of month from here:

May 21, 1994.

Let the current position is before the beginning "M". The following code will save the day of month value into the **strDay** variable:

```
skip_to_chars("0-9");  
begin_var;  
skip_chars("0-9");  
string strDay = end_var;
```

go_start

```
void go_start;
```

If an area of the document is already selected using the **select_area** function then the **go_start** function will reset the cursor to the beginning of the currently selected area. If the **go_start** function is used within the context of a call to select a new area, then the **go_start** function selects the whole document (downloaded in **open_url** function) as an area and moves the cursor to the very beginning of the document.

select_area

```
void select_area(pointer startPtr, pointer finishPtr);
```

This function selects the area in the document to parse. All subsequent operators will operate within the selected area and never leave it. To select the area the programmer should provide two variables of **pointer** type and pass them to **select_area**. The **pointer** type variable is declared as follows:

```
pointer myPtrName
{
    // insert some operations to move cursor to required position in text
    skip_to_text("data starts here");
};
```

The **select_area** function works as follows.

Step 1. It executes the code block defined in the first pointer (typically the first function in the first pointer is **go_start**, see below) and remembers the cursor position when it stops (position 1).

Step 2. Then it executes the second pointer from the position remaining after the first pointer and remembers the cursor position when it stops (position 2).

Step 3. It temporarily bounds the document between position 1 and position 2 and moves the cursor to the position 1.

Example. The document is as follows:

```
I do not like Green Eggs and Ham
I do not like them
Sam, I am

I do not like them here or there
I do not like them anywhere
I do not like them in a boat
I would not, could not, with a goat

I will not eat them in the rain
I do not like them on a train
I do not like them in a box
I will not eat them with a fox

I do not like them in a house
I would not, could not, with a mouse
I do not like Green Eggs and Ham
I do not like them
Sam, I am

Green Eggs and Ham
Green Eggs and Ham
Don't like Green Eggs and Ham
```

The following code will bound the document between the end of the first verse and the end of the second verse:

```
pointer myStartVerse2Ptr
{
    go_start;
    skip_to_text("Sam, I am", true);
};

pointer myFinish Verse2Ptr
```

```

{
    skip_to_text("with a goat", true);
};

select_area(myStart Verse2Ptr, myFinish Verse2Ptr);

```

skip_tags

```
void skip_tags;
```

This function skips all subsequent HTML tags and spaces from the current cursor position and moves the cursor to the beginning of the text after HTML tag and all subsequent spaces if any. Spaces between HTML tags are also skipped. The function skips the whole blocks surrounded by <title>, <script> and <style> tag pairs. The function moves the cursor forward while it finds spaces and HTML tags.

Important: before the call to **skip_tags** function the cursor should not be inside the HTML tag or any text except spaces.

skip_to_tags

```
void skip_to_tags;
```

This function skips the text from current cursor position to the beginning of the HTML tag ("<" character) if any. If no tag is found then the cursor moves to the end of the document/area.

skip_to_text_between_tags

```
string skip_to_text_between_tags(string strText, bool bCaseSensitive);
```

This function looks for text surrounded by HTML tags. The search is either case sensitive or insensitive according to **bCaseSensitive** argument. If the fragment is found, the cursor moves to the position before the first HTML tag after the found text fragment, and the function returns a nonempty string ("."). If the fragment is not found then the cursor remains in the old position and the function returns an empty string. The spaces (and also tabs and line breaks) on the beginning and on the end of the found text are not taken into account.

Important: see the comment to **skip_to_tags** function.

Example:

The document is as follows:

```

<html>
<body>
<font color="red">

    This is a TEXT.

</font>
</body>
</html>

```

Let the cursor be situated before the "<body>" tag.

```

string strFound = "no";

if (skip_to_text_between_tags("this is a text.", false))
    strFound = "yes";

```

The **strFound** variable will get a value **"yes"** and the cursor will move to the position before the "" tag.

skip_to_text

```
void skip_to_text(string strText, bool bCaseSensitive);
```

This function looks for given text from current cursor position. The search is either case sensitive or insensitive according to **bCaseSensitive** argument. If the text is found then the cursor moves to the END of the found fragment, else it doesn't move.

skip_chars

```
void skip_chars(string strCharset);
```

This function skips the symbols of given character set from the current cursor position. The character set is defined in Perl-style: symbol intervals ("A-Z"), symbols enumeration ("QWERTY345") and the negation ("^"). Examples:

"A-Z" – capital letters of the Latin alphabet.

"0-9" – digits.

"a-z0-9" – small letters and digits.

"^abc" – all symbols except "a", "b" and "c".

If the set contains a minus ("-") character the symbol should be preceded with a DOUBLE backslash symbol: "\\-". If the set contains the backslash symbol, it should be copied 4 times: "\\\\".

This function moves the cursor forward until it reaches the character NOT belonging to the given set. The function stops before this character or at the end of the document/area.

skip_to_chars

```
void skip_to_chars(string strCharset);
```

This function moves the cursor forward until it reaches the character belonging to the given set. The character set is defined as in the **skip_chars** function. i.e. **skip_to_chars("0-9")** is the equivalent to **skip_chars("^0-9")**.

get_file_name

```
string get_file_name(string strPath);
```

This function extracts the file name without extension from the full path or URL. Examples:

```
string str = get_file_name("http://aaa.bbb.ccc/dir/subdir/1234.html");
```

The **str** variable will get the value **"1234"**.

```
string str = get_file_name("c:\\windows\\win.ini");
```

The **str** variable will get the value **"win"**.

If the function cannot extract the file name it returns an empty string.

is_today

string is_today (string strDay);

The function handles the **strDay** parameter of one of two possible formats: 1) an English name of the day of week (case is not important) or 2) a decimal number with the day of month.

If the day of week or the day of month coincides with the current date, the function returns a nonempty string ("."), otherwise it returns an empty string.

Example: today is Tuesday, 8th of February, 2005.

```
is_today("wednesday");
```

Returns empty string. And both

```
is_today("8");
```

and

```
is_today("tuesday");
```

will return ".".

get_day

string get_day(string strDayIndex);

The function returns the day of the YEAR from today by the given number of days. The number of days is given by the length of the **strDayIndex** parameter.

Example: let today is Tuesday, 8th of February, 2005. The day of the year is $8 + 31 = 39$. The function call

```
get_day("...");
```

will return "42".

IsImageURLEnabled

string IsImageURLEnabled (string strAutomaticImageNumber);

The function handles the **strAutomaticImageNumber** parameter as a decimal number with the automatic image number.

See "imageURLx" variable below.

The "imageURLx" variable enables the script to automatically configure downloadable images/maps for a specific location based on the weather template. If the user decides to disable this image in PW Options, then the "IsImageURLEnabled" function allows the script to decide at runtime whether to process a section of code – for example, to avoid downloading another URL to access the image URL.

```
IsImageURLEnabled("0");
```

returns empty string if the previously added automatic image URL has been disabled. If it is still enabled, then it returns ".".

add_day

```
void add_day;
```


































The function doesn't take arguments but requires several local variables described below. The variables should be visible in the scope of this function.

Variable	Description
month	the month. Not required (if no such variable present then the days/months are added subsequently from today to the future).
dayOfMonth	a day of month. Not required (if no such variable present then the days are added subsequently from today to the future).
hour	the hour of the day. Not required (if no such variable present then the hour for weather is not specified and the weather is considered as a whole day weather).
taf	the universal packed weather information (see http://www.hut.fi/u/stoivane/metar/#presentWeather) which contains the sky condition, precipitation, wind speed and direction. Not required (if no taf variable then the sky and windXXX variables are used).
sky	the sky condition. Must be an integer number between 0 and 30. The possible values for the sky variable and their meanings can be found below.
skyTextID	the sky condition. Must be an integer number. The possible values for the skyTextID variable and their meanings can be found below. Note that if the template does not provide this variable, it is automatically inferred from the sky variable above.
tmpr (tmprHi and tmprLo)	the temperature (an integer number). If there is no tmpr variable then tmprHi and tmprLo are used (highest and lowest temperature).
tmprUnits	the temperature units. Must be one of these values: "C" (Celsius), "F" (Fahrenheit) and "K" (Kelvin) (case is not important).
dewPoint	the dew point temperature (related to humidity)
humidity	the humidity % (related to dewPoint)
precipitation	the precipitation probability %
visibility	the visibility.
visibilityUnits	The visibility units (feet, inches, nautical miles, millimeters, meters, miles, kilometers)
pressure	the barometric pressure.
pressureUnits	the barometric pressure units. Must be one of these values: " hPa ", " mm " (mmHg), " mb " (mBar) (case is not important).
windSpeed	the wind speed.
windGustSpeed	the maximum wind speed.
windSpeedUnits	the wind speed units. Must be one of these values: " kt " (knots), " mps " (meters per second), " kph " (kilometers per hour), " mph " (miles per hour) (case is not important).
windDirection	the wind direction. Can be either an angle (0-360) or a letter

Variable	Description
	abbreviation: N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, NNW (case is not important).
userText0 - userText9	user text (0-9)
alertText0 - alertText3	alert text (0-3).
forecastUserText	forecast data specific user text.
rainfallUnits	The received rainfall amount (historical) units (feet, inches, nautical miles, millimeters, meters, miles, kilometers)
rainfall	The received rainfall amount (historical)
rainfallDays	The number of days with precipitation ≥ 1 mm
sunshineHours	The average number of hours of sunshine per day (historical)
uvIndex	The UV (sun strength) index
imageURL0-3	If the weather template is able to automatically extract suitable weather image/maps URL/addresses, then setting these variables will cause PocketWeather to automatically update the URL, and download these images. See " IsImageURLEnabled "

Sky Conditions

The following table describes the 31 sky conditions which can be set using the **sky** variable described above. If the **hour** variable is specified and is after dark any sky conditions will automatically be converted to their night equivalent if a day value has been specified.

Index	Sky description	Icon	Index	Sky description	Icon
0	Unknown		21	Haze	
1,2	Rain_And_Wind		22	Smoke	
3,4,17	Thunderstorm		23	Windy	
5	Rain_And_Snow_Mix		24	Very_Windy	
6,18	Sprinkles		25	Frigid	
8	Freezing_Drizzle		26	Cloudy	
9	Drizzle		27	Night_Cloudy	
10	Freezing_Rain		28	80_Clouds	
11	Light_Rain		29	Night_80_Clouds	
12	Rain		30	50_Clouds	
13	Light_Snow		31	Night_50_Clouds	
14	Medium_Snow		32	Clear	
15	Snowing		33	Night_Clear	
16	Snow		34	High_Cloud	
19	Dust		36	Hot	
20	Fog		37	Scattered Thunderstorms	
			39	Scattered Rain	

SkyTextID Conditions Text

The following table describes the sky conditions which can be set using the *skyTextID* variable described above.

Index	Sky description	Index	Sky description	Index	Sky description
0	Unknown	64	Isolated Thunderstorms/Wind	146	AM Light Snow/Wind
1	Cloudy	65	Rain/Snow	147	Scattered Snow Showers/Wind
3	Mostly Cloudy	66	Scattered Thunderstorms/Wind	150	PM Light Rain/Wind
4	Partly Cloudy	67	Am Showers/Wind	152	AM Light Wintry Mix
11	Clear	70	Scattered Snow Showers	153	PM Light Snow/Wind
13	Light Rain	71	Snow to Ice/Wind	154	Heavy Rain/Wind
14	Showers	73	Light Snow Later	155	PM Snow Showers
15	Showers Early	75	Light/Freezing Rain	158	Snow to Rain/Wind
16	Showers Late	77	Snow to Rain	163	Rain/Sleet
18	Rain	78	Snow Showers Early	164	PM Light Rain/Ice
19	AM Showers	80	AM Light Rain	167	AM Snow
20	Scattered Showers	81	PM Light Rain	171	Snow to Ice
21	Few Showers	82	PM Rain	172	Wintry Mix/Wind
22	Mostly Sunny	84	Snow Showers	173	Strong Storms
23	Mostly Clear	85	Rain to Snow	175	PM Light Snow
24	Sunny	86	PM Rain/Snow	178	AM Drizzle
25	Scattered Flurries	88	Few Showers/Wind	183	Snow Showers/Wind Early
26	AM Cloudy / PM Sun	89	PM Snow/Wind	189	Strong Storms/Wind
27	Isolated Thunderstorms	90	Snow/Wind	193	PM Drizzle
28	Scattered Thunderstorms	91	PM Rain/Snow Showers	194	Drizzle
29	PM Showers	92	PM Rain/Snow/Wind	201	AM Light Rain/Wind
30	PM Showers/Wind	93	Rain/Snow Showers/Wind	202	PM Wintry Mix
31	Rain/Snow Showers	94	Rain/Snow/Wind	204	AM Rain/Wind
32	Few Snow Showers	98	Light Snow	223	Wintry Mix to Snow
33	Cloudy/Wind	100	PM Snow	231	Rain
34	Flurries/Wind	101	Few Snow Showers/Wind	236	Ice to Rain
35	Mostly Cloudy/Windy	103	Light Snow/Wind	259	Heavy Rain/Freezing Rain
36	Rain/Thunder	104	Wintry Mix	271	Snow Showers/Windy
37	Partly Cloudy/Windy	105	AM Wintry Mix	988	Partly Cloudy/Windy
38	AM Rain/Snow Showers	106	Heavy Rain/Freezing Rain	989	Light Rain Showers
39	Showers/Wind Late	107	Snow Showers Later	990	Light Rain with

Index	Sky description	Index	Sky description	Index	Sky description
					Thunder
40	Light Rain/Wind	108	AM Light Snow	991	Light Drizzle
41	Showers/Wind	109	Snow	992	Mist
42	Cloudy then Clear	114	Rain/Freezing Rain	993	Smoke
43	Thunderstorms and Showers	118	Thunderstorms/Wind	994	Haze
44	Mostly Sunny/Windy	123	Sprinkles	995	Light Snow Showers
45	Flurries	125	AM Snow Showers	996	Light Snow Showers/Windy
46	Mostly Clear/Wind	126	AM Clouds/PM Sun/Wind	997	Clear
47	Rain/Wind	127	Rain/Snow Showers Later	998	A Few Clouds
49	Scattered Flurry/Wind	128	AM Rain/Snow/Wind	999	Fair
50	Scattered Strong Storms	130	Rain to Snow/Wind	1000	Strong Winds
51	PM Thunderstorms	132	Snow to Wintry Mix	1001	Strong Thunderstorm
52	Thunderstorms Early	133	PM Snow Showers/Wind	1002	Funnel Cloud
53	Thunderstorms	135	Snow and Icon to Rain	1003	Tornado
55	Sunny/Windy	137	Heavy Rain	1004	Tornado in the Vicinity
56	AM Thunderstorms	138	AM Rain/Ice	1110	Tropical Storm
57	Thunderstorms Later	140	Heavy Thunderstorms/Wind	1111	Cat. 1 Hurricane
62	AM Rain	142	Rain/Thunder/Wind	1112	Cat. 2 Hurricane
63	Scattered Showers/Wind	145	Am Snow Showers/Wind	1113	Cat. 3 Hurricane
				1114	Cat. 4 Hurricane
				1115	Cat. 5 Hurricane

CWF Development Tool

A simple PC tool is provided to assist CWF development. An existing CWF file can be selected in the tool using a browse button.

Edit

An Edit button opens the selected CWF file into an Edit window where the CWF definition can be edited and saved.

Save, Save As

Save and Save As buttons enable a modified CWF file to be saved.

Run

A Run button causes the CWF to be compiled and if successful to be executed.

Close

A Close button causes the CWF tool to exit.

Compilation

When the CWF is compiled any errors are reported in a report window as an error message and the line at which the error occurs. The line containing the error (or the following line) is highlighted in the edit window. The following errors are possible:

Message
Unrecognized symbol found
" ," expected
"{" expected
"}" expected
"(" expected
")" expected
"[" expected
"]" expected
Boolean constant expected
String constant expected
Identifier expected
Undeclared identifier
" ," expected
Syntax error

Debugging

In order to debug execution of the CWF to parse the HTML data, debug information is also directed into the report window. Debug information is recorded for most functions:

- String variable assignment.
- Each **open_url** function call.
- HTML text navigation functions. The 50 characters following the current cursor position in the HTML text are shown.
- Each **add_day** function call.

Below is an extract from a typical example of a debug log from a successful CWF.

```
open_url("http://www.bbc.co.uk/weather/5day.shtml?id=3203", "GET")
sky = "."
foundToday = ""
tmprUnits = "C"
windSpeedUnits = "mph"
pressureUnits = "mb"
dayOfWeek = "Wednesday"
foundToday = "."
sky = "3"
sky = "4"
tmprHi = "2"
tmprLo = "0"
windDirection = "NE"
windSpeed = "13"
pressure = "1016"
humidity = "74"
add_day
...

dayOfWeek = "Sunday"
sky = "3"
sky = "4"
tmprHi = "5"
tmprLo = "-2"
windDirection = "NE"
windSpeed = "12"
pressure = "1026"
humidity = "68"
add_day
dayOfWeek = ""
sunrise = ""
sunset = ""
sky = ""
sky = ""
sky = "0"
tmprHi = ""
tmprLo = ""
windDirection = ""
windSpeed = ""
pressure = ""
humidity = ""
```