CAPLIN

# DS4C 6.0

## DataSource For C
## Configuration Syntax Reference

December 2012

# Contents



 

# 1 Preface

## 1.1 What this document contains

This document describes the syntax of the language that is used to configure DataSource applications that have been built using Caplin's DataSource for C API. Such applications include Caplin Liberator, Caplin Transformer, and Integration Adapters that use the API.

### About Caplin document formats

This document is supplied in two formats:

- Portable document format (*.PDF* file), which you can read on-line using a suitable PDF reader such as Adobe Reader®. This version of the document is formatted as a printable manual; you can print it from the PDF reader.
- Web pages (*.HTML* files), which you can read on-line using a web browser. To read the web version of the document, navigate to the *HTMLDoc* folder and open the file *index.html*.

#### For the best reading experience

On the machine where your browser or PDF reader runs, install the following Microsoft Windows® fonts: Arial, Courier New, Times New Roman, Tahoma. You must have a suitable Microsoft license to use these fonts.

## 1.2 Who should read this document

This document is intended for System Administrators who need to install, configure, and manage DataSource for C applications. DataSource for C applications included Caplin Liberator, Caplin Transformer, and Integration Adapters that use the API.

## 1.3 Related documents

- **Caplin DataSource Overview**

  A technical overview of Caplin DataSource.

- **DataSource for C API Documentation**

  The API reference documentation that allows application developers to write DataSource for C applications.

## 1.4 Typographical conventions

The following typographical conventions are used to identify particular elements within the text.

| *Type* | *Uses* |
|---|---|
| **aMethod** | Function or method name |

| | |
|---|---|
| *aParameter* | Parameter or variable name |
| */AFolder/Afile.txt* | File names, folders and directories |
| `Some code;` | Program output and code examples |
| The `value=10` attribute is... | Code fragment in line with normal text |
| Some text in a dialog box | Dialog box output |
| `Something typed in` | User input – things you type at the computer keyboard |
| **Glossary term** | Items that appear in the “Glossary of terms and acronyms” |
| **XYZ Product Overview** | Document name |
| ◆ | Information bullet point |
| ■ | Action bullet point – an action you should perform |

**Note:** Important Notes are enclosed within a box like this.
Please pay particular attention to these points to ensure proper configuration and operation of the solution.

**Tip:** Useful information is enclosed within a box like this.
Use these points to find out where to get more help on a topic.

Information about the applicability of a section is enclosed in a box like this.
For example: “This section only applies to version 1.3 of the product.”

## 1.5 Feedback

Customer feedback can only improve the quality of our product documentation, and we would welcome any comments, criticisms or suggestions you may have regarding this document.

Visit our feedback web page at https://support.caplin.com/documentfeedback/.

## 1.6 Acknowledgments

# 2 Introduction

**DataSource for C applications** can be configured by settings stored in one or more configuration files, or by settings stored in a database and supplied to the **DataSource application** by a configuration script.

This document describes:

- The syntax of the different types of configuration item used to configure a DataSource for C application.
- How variables, conditionals, and **macros** can be used in a configuration file.
- How configuration settings supplied by a web server or configuration script can be used to configure a DataSource for C application.

For a description of the configuration items that you can use to configure a particular DataSource for C application (such as **Caplin Liberator**, **Caplin Transformer** or an **Integration Adapter**), refer the Administration Guide for the application that you want to configure. For a description of the DataSource communications infrastructure and how it links Caplin components together, see the **Caplin DataSource Overview** document.

# 3 Types of configuration item

Four different types of configuration item can be used to configure a DataSource for C application:

- Boolean configuration items 5
- Single value configuration items 5
- Multi value configuration items 6
- Configuration items with nested options 6

Some of these configuration items allow you to specify a list of one or more values (see Lists 8).

## 3.1 Boolean configuration items

A boolean configuration item does not take any value, but is set to true if present in the DataSource configuration, and to false (the default) if omitted from the configuration. An example of a boolean configuration item is **latency-chain-enable**, which enables or disables latency chaining.

**Example**

```
latency-chain-enable
```

In this case latency chaining is enabled.

In the reference sections of the Administration Guide for the DataSource application, the syntax definition specifies further information about boolean configuration items, as shown in the following table.

| Type | Default | Permitted values |
|---|---|---|
| boolean | false | True or false, and a description of the effect of each value (such as true to enable or false to disable). |

## 3.2 Single value configuration items

Single value configuration items set something to a single value. An example of this type of configuration item is **datasrc-port**, which specifies the network port that the application listens on for DataSource messages and connection requests.

**Example**

```
datasrc-port  22001
```

In this case the listening port is set to `22001`.

If present in the configuration file, a single value configuration item must define the value to be assigned. If not present in the configuration file, the configuration item is assigned a default value.

In the reference sections of the Administration Guide for the DataSource application, the syntax definition specifies further information about single value configuration items, as shown in the following table.

| Type | Default | Permitted values |
|---|---|---|
| The type of value, such as string or integer. | The default value, or <no default> if there is no default value. | Restrictions on the permitted values for this configuration item. |

## 3.3 Multi value configuration items

Multi value configuration items set the value of two or more related items. An example of this type of configuration item is **add-field**, which maps a field name (a string) to a field number (an integer).

**Example**
```
add-field  Bid 22
```

In this case the `Bid` field is mapped to field number `22`.

If a multi value configuration item is present in the configuration file, values must be defined for all related items that are not optional (such as the field name and field number in the example above). If a related item is described as optional in the Administration Guide (such as optional field flags data), the value for that related item can be omitted from the configuration. Values are assigned according to the position of the value in the list of ordered values. If the configuration item is not present in the configuration file, related items are assigned default values.

In the reference sections of the Administration Guide for the DataSource application, named position indicators in the syntax definition indicate how values of related items must be ordered. In the example **add-field** configuration item above, the syntax would look something like this.

**Example Syntax for the add-field configuration item**
```
add-field  FieldName  FieldNumber
```

Further information about the items that these named position indicators represent are shown in a table like this.

**Options**

| Name | Type | Default | Description |
|---|---|---|---|
| The name of the position indicator, such as `FieldName`. | The type of value, such as string or integer. | The default value, or <no default> if there is no default value. | Describes the item represented by the named position indicator, and defines any restrictions on its value. |

## 3.4 Configuration items with nested options

This type of configuration item takes nested options, where each nested option can be any one of the four types of configuration item (boolean, single value, multi value, and nested). An example of this type of configuration item is **add-peer**, which specifies options for connecting to a remote DataSource application (a **DataSource peer**).

**Example**

```
add-peer
        addr            liberator.example.com
        port            25000
end-peer
```

In this case the address of the remote DataSource is set to `liberator.example.com`, and the port that it listens on to `25000`.

In the reference sections of the Administration Guide for the DataSource application, the syntax definition specifies the configuration items that can be nested. In the example **add-peer** configuration item above, the syntax would look something like this.

**Example Syntax for the add-peer configuration item**

```
add-peer
        addr            <value>
        port            <value>
end-peer
```

This syntax indicates that if the **addr** and **port** configuration items are present in the configuration, then a value must be defined for each of these items. The angle brackets indicate that `<value>` is only a place holder for the value, and not the value itself.

The next example shows the syntax for other types of nested configuration item.

**Syntax for nested configuration items**

```
add-something
        boolean-type
        single-value-type       <value>
        multi-value-type        <values>
end-something
```

The syntax shows that the **add-something** configuration item has three nested options, each type identified by the values they take:

**boolean-type** Set to true if present in the configuration, otherwise false.

**single-value-type** Sets a single value if present in the configuration, otherwise takes a default value.

**multi-value-type** Sets multiple values if present in the configuration, otherwise takes default values.

## 3.5 Lists

Some configuration items allow you to specify a list of one or more values, where each value is the same type. An example is **addr**, which defines a list of addresses that the DataSource application will attempt to connect to.

In the reference sections of the Administration Guide for the DataSource application, the syntax definition specifies the types that can be listed, as shown in the following table.

| List Type | Description | Example |
|---|---|---|
| string list | A space separated list (array) of one or more strings. | A space separated list of IP addresses.<br>127.0.0.0 192.255.129.1 |
| integer list | A space separated list (array) of one or more integers. | A space separated list of port numbers.<br>22001 22002 |

# 4 Configuration Variables

Two kinds of variable can be used to configure a DataSource for C application: defined variables and environment variables.

## 4.1 Defined variables

A configuration variable is defined using the **define** configuration directive, and undefined using the **undefine** directive. The scope of the variable is the configuration file in which it is defined, and any files that include this configuration file using the **include-file** directive
(see Including multiple configuration files 17).

**Notation: defining and undefining a variable**
```
define VARIABLE value

undefine VARIABLE
```

In the notation shown above, `VARIABLE` is the name of the variable and `value` is the value that it takes. The name of the variable can contain alphanumeric characters and the characters "." and "_" (without the quotation marks), and by convention is always uppercase. If `value` is a string that contains spaces, the string must be enclosed in single or double quotation marks.

The following notation shows how a variable is used to set the value of a configuration item.

**Notation: using a variable in the configuration**
```
config-item ${VARIABLE}
```

In the notation shown above, **config-item** is the name of the configuration item, and `VARIABLE` is the name of the variable that sets the value of the configuration item.

When a variable is used to set the value of a configuration item, the name of the variable must be enclosed in curly braces and preceded by the `$` character. If an undefined variable is used in the configuration, the value of the configuration item is set to an empty string and a warning is logged to the console and log file.

The following example defines a variable, and sets the value of a configuration item using that variable.

**Example: defining and using a variable**
```
define HTTP_PORT 8080

http-port ${HTTP_PORT}
```

In this case, the value of the **http-port** configuration item is set to `8080` (using the `HTTP_PORT` variable).

Variables can also be concatenated with surrounding text and other variables.

**Example: concatenating a variable with surrounding text**
```
http-port 1${HTTP_PORT}
```

In this case, the value of the **http-port** configuration item is set to `18080`.

 

## Pre-defined Variables

The following variables are pre-defined by the **DataSource for C library**:

**Pre-defined variables**

| Name | Description |
|---|---|
| `APP_NAME` | The name of the DataSource for C application. Examples are `transformer` and `rttpd`. |
| `CPU` | The type of processor. Examples are `i686` and `sparc`. |
| `HOME_DIR` | The home directory of the process owner. |
| `HOST_NAME` | The name of the host machine. |
| `MAJOR_VERSION` | The major version of the DataSource for C library (for example, 5) |
| `MINOR_VERSION` | The minor version of the DataSource for C library (for example, 0) |
| `OS` | The name of the operating system that the DataSource for C library was built for.<br>Examples are `linux-gnu` and `solaris2.10`. |
| `OS_TRIPLET` | The GNU configuration triplet (of the form `cpu-manufacturer-operating_system`).<br>Examples are `i686-pc-linux-gnu` and `sparc-sun-solaris2.10`. |

## Using complex expressions to define a variable

Configuration variables can be defined using complex expressions that are evaluated by an arithmetic expressions calculator. The expressions calculator is invoked using the following notation:

**Notation to invoke the expressions calculator**

```
${EVAL:expression}
```

In the notation shown above, `expression` is a list of items and arithmetic operators that the expressions calculator evaluates. Each item in `expression` must have a numeric value, and the valid arithmetic operators are `+` (addition), `-` (subtraction), `/` (division), and `*` (multiplication). Parenthesis can also be used to group parts of an `expression` (see Appendix: Supported mathematical operators 22 for examples of operator precedence).

 

The following example uses the expressions calculator to add `80` to the value of the defined variable `PORT_BASE`.

**Expressions calculator: setting a port number**

```
define PORT_BASE 20000
http-port ${EVAL:${PORT_BASE} + 80}
```

In this case the `expression` evaluated is `${PORT_BASE} + 80`. Because `PORT_BASE` has the value 20000, the **http-port** configuration item (the HTTP port number) is set to 20080.

The @ operator allows a configuration item to be used inside an `expression`. The next example sets the HTTP port number to 8080, and then uses this setting to calculate and set the HTTPS port number.

**Expressions calculator: evaluating a configuration item**

```
http-port 8080
https-port ${EVAL:@http-port + 1}
```

In this case **http-port** (the HTTP port number) is set to 8080, and **https-port** (the HTTPS port number) is set to `8081`.

## 4.2 Environment variables

Environment variables can be used to set the value of a configuration item.

**Notation: using an environment variable in the configuration**

```
config-item ${ENV:VARIABLE}
```

In the notation shown above, **config-item** is the name of the configuration item and `VARIABLE` is the name of the environment variable that sets the value of the configuration item.

The name of the environment variable must be preceded by the text `ENV:` and enclosed in curly braces, and the braces must be preceded by the `$` character. If the environment variable has not been defined, the value of the configuration item is set to an empty string and a warning is logged to the console and log file.

The following example sets the value of a configuration item using an environment variable.

**Example: defining and using an environment variable**

```
runtime-user ${ENV:USER}
```

In this case, the value of **runtime-user** is set to the value of the `USER` environment variable.

# 5 Conditional flow control

DataSource for C libraries support conditional flow control constructions that have `if`/`else`/`elseif`/`endif` statements in the DataSource configuration.

**Flow control construction**

```
if condition1
...configuration 1
elseif condition2
...configuration 2
else
...configuration 3
endif
```

In the notation shown above, `condition1` and `condition2` are conditions that evaluate to true or false.

- If `condition1` evaluates to true, `configuration 1` is applied, otherwise the `elseif` condition is tested.
- If the `elseif` condition is tested and `condition2` evaluates to true, `configuration 2` is applied.
- If `condition1` and `condition2` evaluate to false, `configuration 3` is applied.

The `elseif` and `else` statements are optional and the `endif` statement ends the conditional flow control. There can be more than one `elseif` statement inside an `if`/`endif` construct, but only one `else` statement.

Each condition can have a simple or complex form, depending on whether the condition is part of a simple or complex conditional statement.

## 5.1 Simple conditional statements

A simple conditional statement is shown below.

**Simple conditional statement**
```
if test value
```

In a simple conditional statement, `test` is a data item to test and `value` is the value to test against.

By default, the following data items can be tested:

**Data items available to test**

| Name | Description |
|---|---|
| application | The name of the application<br>(set using the **application-name** configuration item). |
| hostname | The name of the host machine. |
| os | The name of the running operating system (such as `linux-gnu`, `linux-EL5-gnu`, and `solaris2.10`). |

A DataSource for C application can specify other data items to test (see **ds_config_set_test()** in the **DataSource for C API Documentation**), but no Caplin products do so.

### Example

The following example sets **http-port** to `8080` if the application is `rttp`, and to `8082` otherwise.

**Example of a simple conditional statement**
```
if application rttp
  http-port 8080
else
  http-port 8082
endif
```

## 5.2 Complex conditional statements

A complex conditional statement is shown below.

**Conditional statement (complex form)**

```
if test operator value
```

In a complex conditional statement, `test` is a data item to test, `value` is the value to test against, and `operator` is a comparison operator used to evaluate the condition.

The following operators (shown separated by commas) can be used inside a complex conditional flow control statement (see Appendix: Supported mathematical operators 22 for further information):

- Arithmetic operators: `*`, `-`, `/`, `+`
- Comparison operators: `<`, `>`, `>=`, `<=`, `==`, `!=`
- Parenthesis: `( )`
- Logical operators: `and`, `&&`, `or`, `||`
- The `@` operator identifies a configuration item.

If `value` is a string that contains spaces, the string must be enclosed in single or double quotation marks. This means that the following `if` statements are equivalent:

```
if ${HOST_NAME} == "myserver"
```

```
if ${HOST_NAME} == 'myserver'
```

```
if ${HOST_NAME} == myserver
```

Conditions that evaluate to `0` are false, and conditions that evaluate to any other number are true. Conditions that evaluate to a string are false.

### Examples

The `@` operator allows a configuration item to be used in a complex conditional statement.

**Example: using the @ operator**

```
if @http-port == 80
   https-port 443
endif
```

In this case the **https-port** is only set to `443` if the **http-port** is set to `80`.

The `@` operator can also be used with a boolean configuration item to construct the condition that is evaluated.

**Example: using the @ operator with a boolean configuration item**

```
if @https-enable
   https-port 443
endif
```

In this case the **https-port** is only set if HTTPS enabled (that is, if the boolean configuration item **https-enable** evaluates to true).

The next example shows how logical operators can be used to combine conditions in a flow control statement.

**Example: combining conditions using logical operators**

```
if ${HTTPS} == 1 and ${HOST_NAME} == myserver
   https-enable
   https-port 443
endif
```

In this case the configuration between the `if` and `endif` statements is only applied if the configuration variable `HTTP` is set to `1` **and** the configuration variable `HOST_NAME` is set to `myserver`.

# 6 Using macros in a configuration

Macros can make product configuration easier and less prone to errors. Although no predefined macros are provided with the **DataSource for C SDK**, the DataSource for C library allows you to define and use your own macros in a product configuration.

A macro definition starts with a `defmacro` directive and ends with an `endmacro` directive, and can span several lines. The following example defines a macro called **active_peer** that can be invoked later in the configuration.

**Example macro definition**

```
defmacro active_peer(name, id)
add-peer
     addr  127.0.0.1
     port  25000
     local-id ${id}
     local-type active
     local-name ${name}
     label ${name}
end-peer
endmacro
```

In this example the macro takes two arguments (`name` and `id`) that are used in the definition of an **add-peer** configuration item. When the macro is invoked, a value must be passed in for each of the required arguments.

**Invoking the example macro**

```
active_peer("SSLsrc", 10)
```

Invoking **active-peer** with the arguments shown above adds a DataSource peer to the configuration, where **local-id** is set to `10`, **local-name** to "`SSLsrc`", and **label** to "`SSLsrc`". Invoking **active-peer** with different arguments would add a different DataSource peer.

## Macro restrictions and limitations

When a macro is defined, the following restrictions and limitations apply:

- The scope of the macro is the configuration file in which it is defined, and files that include this configuration file using the **include-file** directive (see Including multiple configuration files 17).
- Arguments to the macro are treated like defined variables, and are accessed in the macro definition by enclosing the name of the argument in curly braces preceded by the `$` character.
- A macro can be invoked inside another macro definition.
- A macro must not invoke itself.
- If a macro is invoked with missing arguments, an empty string replaces the value of each missing argument.

# 7 Including multiple configuration files

The **include-file** directive specifies that another configuration file has configuration settings for the DataSource for C application. The * wildcard can be used in the relative path of the included file to include multiple configuration files.

**Example**

```
include-file FX/*.conf
```

In this case, configuration settings from all files in the *FX* directory that have a *.conf* file extension are included in the configuration.

# 8 Obtaining configuration from a webserver

The configuration for a DataSource for C application can be obtained from a web server by specifying the URL of the required configuration file in an **include-file** directive.

**Specifying the URL of a configuration file**

```
include-file http://configurationserver/rttpd.conf
```

The example above specifies that configuration in the file *rttpd.conf* can be obtained from the web server with the domain name *configurationserver* using the *http* protocol. Note that the protocol must be HTTP. If a retrieval utility is not specified in the configuration that contains the **include-file** directive, **wget** is used to retrieve the configuration.

A retrieval utility can be specified using the **http-download-command** directive or as part of the **include-file** directive. In either case the specified utility must send the configuration data to **standard output** (**stdout**).

The following example sets **wget** as the retrieval utility (the default utility if the **http-download-command** directive is omitted).

**Obtaining a configuration using wget**

```
http-download-command "wget -nv -O -"

include-file http://configurationserver/rttpd.conf
```

The next example sets **curl** as the retrieval utility.

**Setting curl as the retrieval utility**

```
http-download-command "curl -s"

include-file http://configurationserver/rttpd.conf
```

The alternative notation, which specifies the retrieval utility (in this case **curl**) as part of the **include-file** directive, is shown below.

**Setting curl as the retrieval utility: alternative notation**

```
include-file "|curl -s http://configurationserver/rttpd.conf"
```

If you use this alternative notation, the argument to **include-file** must be enclosed in quotation marks, and the pipe character ( | ) must precede the name of the retrieval utility. This instructs the DataSource for C library to execute the named utility rather than read configuration data from a file.

# 9 Obtaining configuration from a script

The configuration for a DataSource for C application can be generated dynamically and obtained from a script by specifying the script in an **include-file** directive. A typical example would be when a script queries a database to obtain the configuration.

**Specifying a script that returns configuration**

```
include-file "|query_rttpd_configuration.sh"
```

The example above specifies that configuration can be obtained from the *query_rttpd_configuration.sh* script.

The argument to **include-file** must be enclosed in quotation marks and the pipe character ( | ) must precede the name of the script. This instructs the DataSource for C library to execute the script rather than read configuration data from a file.

When configuration is obtained from a script, the script must write the configuration data to **standard output** (**stdout**).

# 10 Enabling a macro preprocessor

DataSource for C configuration files can be processed by a macro preprocessor before the content of the file is parsed by the DataSource for C library. A macro preprocessor is specified by passing the *--preprocessor-binary* command line option to the DataSource for C application on startup.

**Tip:** If a macro preprocessor is specified, it runs *before* macros defined in the configuration file are processed by the DataSource for C library (see Using macros in a configuration 16).

The specified macro preprocessor must take the name of the configuration file as an input parameter and print the processed configuration to standard output. The **GNU M4** macro preprocessor meets this requirement.

**Note:** The **CPP** C preprocessor also meets this requirement, but is not recommended as it corrupts the content of configuration files.

Non GNU M4 preprocessors are not recommended in Caplin Trader installations, because Caplin Trader uses functionality that is only present in GNU M4 preprocessors. Non GNU M4 preprocessors are also likely to cause problems in Oracle Solaris® systems.

The *properties* files of Java components are not processed by the macro preprocessor.

## Caplin DataSource for C products – enabling a macro preprocessor

All Caplin products built with DataSource for C, such as Liberator and Transformer, are provided with a startup script to start the application. Each startup script contains a `PREPROCESSOR` environment variable that sets the value of the *--preprocessor-binary* command line option when the application starts. The `PREPROCESSOR` environment variable can either be set to the path of the macro preprocessor binary, or to a script that starts the macro preprocessor.

Some macro preprocessors have built in commands that must be disabled. An example is the GNU M4 macro preprocessor, which removes all instances of the word 'format' from the files that it processes. This causes a problem with the configuration of Caplin Transformer, which has a module called **format**. If the word 'format' is removed from the Transformer configuration, an invalid Transformer licence is reported and Transformer will fail to start.

To overcome this problem, the `PREPROCESSOR` environment variable (in the application startup script) can be set to a script that starts the macro preprocessor but disables the unwanted 'format' command.

**Setting the PREPROCESSOR environment variable in the application startup script**

```
PREPROCESSOR=/usr/local/bin/preprocess.sh
```

In this case the PREPROCESSOR environment variable is set to `/usr/local/bin/preprocess.sh`.

The content of `/usr/local/bin/preprocess.sh` is shown below.

**The macro preprocessor startup script**

```
#!/bin/sh
m4 -Uformat $1
```

In this case the script starts the M4 macro preprocessor and disables the 'format' command. A suitable M4 startup script that disables the 'format' command is provided with Caplin Platform installations.

## Custom DataSource for C applications – enabling a macro preprocessor

To specify a macro preprocessor to a custom DataSource for C application that you write, pass the *--preprocessor-binary* command line option to the application on startup. The command line option must specify the path of the macro preprocessor binary, or a custom script that starts the macro preprocessor.

**Command line option specifying a macro preprocessor startup script**

```
--preprocessor-binary=/usr/local/bin/preprocess.sh
```

In this case the *--preprocessor-binary* command line option specifies a macro preprocessor startup script (*/usr/local/bin/preprocess.sh*), and not a macro preprocessor binary.

# 11 Appendix: Supported mathematical operators

The following mathematical operators can be used in a conditional flow control statement.

The operators are listed in order of precedence (operators at the top of the table have higher precedence than operators at the bottom of the table).

**Supported operators (in order of precedence)**

| Operators | Description |
|---|---|
| `()` | Parenthesis (grouping). |
| `*  /` | Multiplication and division. |
| `+  -` | Addition and subtraction |
| `<  <=  >  >=` | Comparisons: less than, less than or equal to, more than, more than or equal to. |
| `== !=` | Comparisons: equal to, not equal to. |
| `&&  and` | Logical AND (`&&` or `and` can be used). |
| `\|\|  or` | Logical OR (`\|\|` or `or` can be used). |

The following table shows some examples of operator use and precedence:

**Operator use and precedence**

| Example | Comment |
|---|---|
| `2 + 3 * 3 = 11` | Multiplication is completed before addition. |
| `(2 + 3) * 3 = 15` | The operation in parenthesis is completed before the multiplication. |
| `if ${HTTPS} == 1 && ${HOST_NAME} == myserver` | True if the variable `HTTPS` is set to `1` **and** the variable `HOST_NAME` is set to `myserver`. |
| `if ${HTTPS} == 1 or ${HOST_NAME} == myserver` | True if the variable `HTTPS` is set to `1` **or** the variable `HOST_NAME` is set to `myserver`. |

For a description of the flow control constructions you can use to configure a DataSource for C application, see Conditional flow control 12.

 

# 12 Glossary of terms and acronyms

This section contains a glossary of terms, abbreviations, and acronyms used in this document.

| Term | Definition |
|---|---|
| **Caplin Liberator** | A financial internet hub that delivers data and messages in real time to and from subscribers over any network. |
| **Caplin Platform** | An integrated suite of software that supports the services and distribution capabilities needed for web trading. It consists of **Caplin Liberator**, **Caplin Transformer**, Caplin KeyMaster, Caplin Director, and Caplin Management Console. |
| **Caplin Transformer** | An event-driven, real-time data transformation engine optimized for web trading services. |
| **Configuration file** | A file containing the initial settings for a software application. |
| **Configuration item** | An entry in a **configuration file** that holds the value of one or more application settings (see Types of configuration item 5). |
| **CPP** | A preprocessor for the C language. |
| **Integration Adapter** | A server application that allows an external system to communicate with the **Caplin Platform**. An Integration Adapter is a **DataSource application**. |
| **DataSource application** | An application that uses the **DataSource API**.<br>**Caplin Liberator**, **Caplin Transformer**, and **Integration Adapters** are all DataSource applications. |
| **DataSource for C application** | A **DataSource application** written in the C programming language using the **DataSource for C SDK**. **Caplin Liberator**, **Caplin Transformer**, and some **Integration Adapters** are DataSource for C applications. |
| **DataSource for C API** | The API provided by the **DataSource for C Library**. |
| **DataSource for C Library** | A software library that allows a DataSource for C application to send messages to other **DataSource applications**. |
| **DataSource for C SDK** | A software development kit that allows you to build custom **DataSource for C** applications. |
| **DataSource peer** | A remote **DataSource application** that is connected to this DataSource application. |
| **GNU** | A project launched in 1984 to develop a Unix-like operating system. |
| **M4** | A general purpose macro preprocessor. |
| **Macro** | A named list of configuration items and macro arguments. A macro is defined once in a configuration file but can be 'called' several times later with values that customize the applied configuration (see Using macros in a configuration 16). |

CAPLIN
THE WEB TRADING TECHNOLOGY COMPANY

## Contact Us

Caplin Systems Ltd

Cutlers Court

115 Houndsditch

London  EC3A 7BR

Telephone: +44 20 7826 9600

**www.caplin.com**