applparms: Application Parameters Library User Guide and Reference Version 1

CML00054-01

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
`www.codemagus.com`

December 15, 2020

Contents

1 Introduction

The `applparms` library, together with the defined user interface, is a library that enables an application to define, manipulate and store the current state of a parameter set. Before the application can start the user has the ability to set and change the values of any of the parameters in the set and the library will not allow the application to start until all the parameters have a valid value. The parameter attributes define what a valid parameter is. To this extent the set of parameters defined by a particular configuration are bound to that application; meaning that each application would have its own set of parameters.

An application that requires the use of a set of parameters and their values will, usually during initialisation, call the `applparms` library open function passing it the name of the application parameter configuration file.

The `applparms` library will then

- Parse and load the configuration of parameters into memory. See section 2 on page 7
- Read a log file of previous changes and set all the parameter values to that of the current saved state.
- Invoke a user interface which will allow the user to set and/or change the value of each parameter. See section 6 on page 31 for the default command line interface and section 7 on page 37 and section 8 on page 39 for information on the tabs and list graphical user interfaces respectively.

The parameter values can then be used by the application during processing in any way; for example one parameter may be the name of the output file the application is to produce.

The library exports various functions (see section 3 on page 18) that the application can call:

- An open function that will instantiate the `applparms` internal structure.
- A close function that will remove the `applparms` internal structure.
- A get function that allows a parameter value to be retrieved.

The library also has facilities (see section 4 on page 23) that the user interface may call:

- Functions for setting and retrieving parameter values.
- A function for setting mark points.
- A function for resetting all parameters back to a previous state using a known mark point or timestamp.
- A function to check how many parameters have not had a value set.

Once the application has completed its processing it should call the `applparms` close function in order to release all resources held by the library.

2 Configuration File Syntax

2.1 Introduction

The configuration file defines all the parameters to be used by the application that names it in the open call. It also defines each parameter's properties and all the general properties such as the user interface.

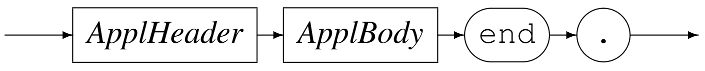
For an example of a configuration file see section 2.3 on page 15.

2.2 Syntax

2.2.1 *ApplSpecification*

The configuration file comprises of comments and the header and body sections and is terminated with the `end` keyword and a full stop.

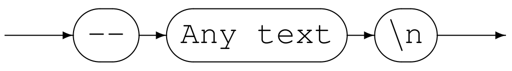
ApplSpecification



2.2.2 *Comments*

Comments begin anywhere with a double dash and continue up to the end of the line.

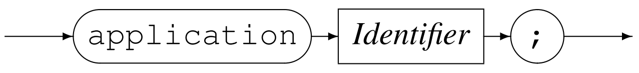
Comments



2.2.3 *ApplHeader*

The header section defines the name of the configuration and must match the actual configuration file name.

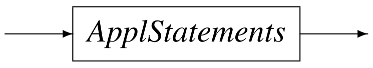
ApplHeader



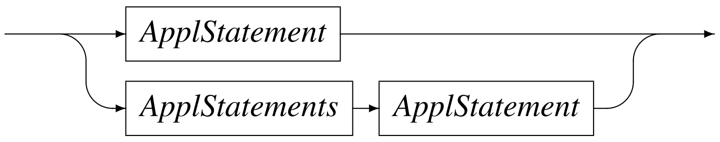
2.2.4 *ApplBody*

The body section is made up of one or more statements.

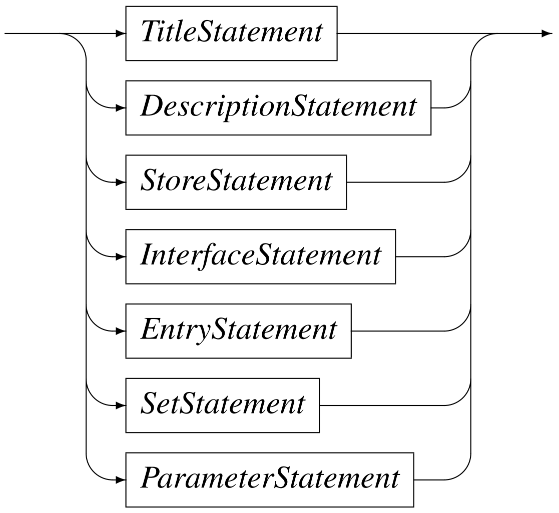
ApplBody



ApplStatements



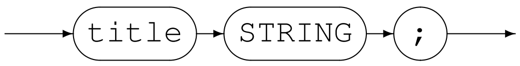
ApplStatement



2.2.5 *TitleStatement*

This sets the title of the configuration. See section 2.2.18 on page 13 for information on writing long strings.

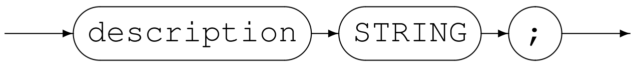
TitleStatement



2.2.6 *DescriptionStatement*

This sets the description of the configuration. See section 2.2.18 on page 13 for information on writing long strings.

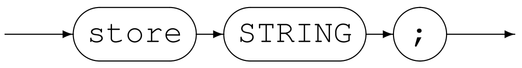
DescriptionStatement



2.2.7 *StoreStatement*

This names the store where the log of updates and mark points will be written to. The log will have the same name as the configuration file with the suffix of `.apd`.

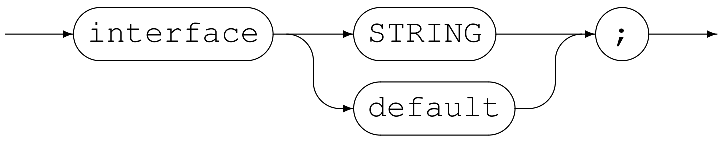
StoreStatement



2.2.8 *InterfaceStatement*

This names the shared program or DLL that will be invoked as the user interface. This must be a *STRING* value unless the keyword `default` is used. The default user interface is a command line program to which the user submits commands via `stdin` and views output via `stdout`. Other user interfaces may, for example on Windows, be a GUI.

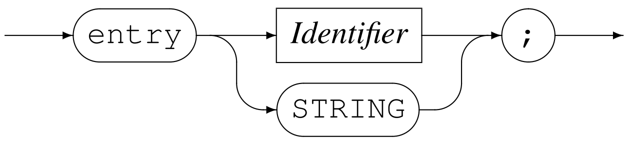
InterfaceStatement



2.2.9 *EntryStatement*

This names the program entry point in the interface program. This is usually of the form `<program_name>_entry`. Where `program_name` is the base name of the interface program name. This option is also usually an *Identifier*, but may be a `STRING` to allow non-standard names or names that are held in an environment variable (these return a `STRING` as shown in section 2.2.18 on page 13.

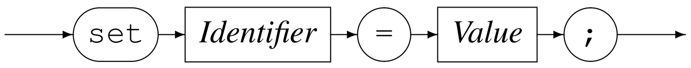
EntryStatement



2.2.10 *SetStatement*

This allows any environment variable to be set for use further down in the configuration file.

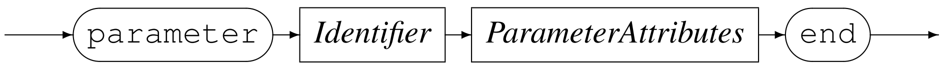
SetStatement



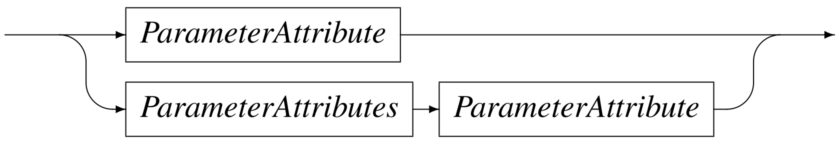
2.2.11 *ParameterStatement*

This defines a parameter and all of its attributes. A parameter may only be named once.

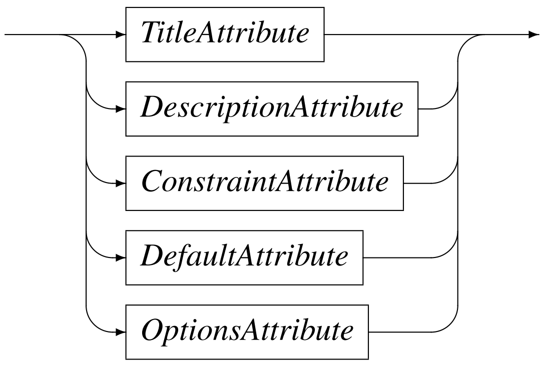
ParameterStatement



ParameterAttributes



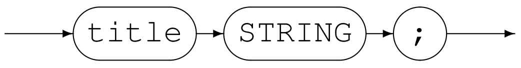
ParameterAttribute



2.2.12 *TitleAttribute*

This defines the title of the parameter. See section 2.2.18 on page 13 for information on writing long strings.

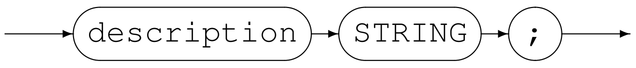
TitleAttribute



2.2.13 *DescriptionAttribute*

This adds a description to the parameter. See section 2.2.18 on page 13 for information on writing long strings.

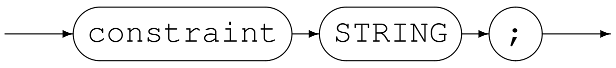
DescriptionAttribute



2.2.14 *ConstraintAttribute*

The constraint pattern constrains the value of the parameter. If the value offered does not match this pattern then the parameter value is not updated. See section 2.2.18 on page 13 for information on writing long strings.

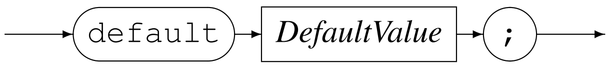
ConstraintAttribute



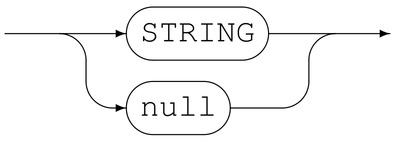
2.2.15 *DefaultAttribute*

This is the value that the parameter is set to initially. This value must conform to all constraints in place for the parameter. To not set an initial value the keyword `NULL` may be used. See section 2.2.18 on page 13 for information on writing long strings.

DefaultAttribute



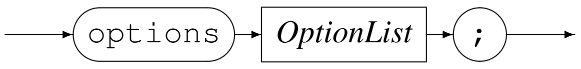
DefaultValue



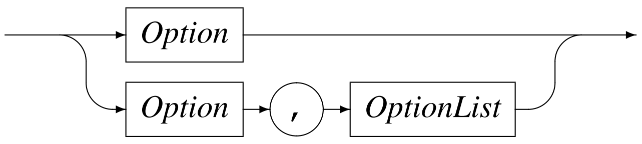
2.2.16 *OptionsAttribute*

The Options attribute sets characteristics of a parameter. This attribute is not required.

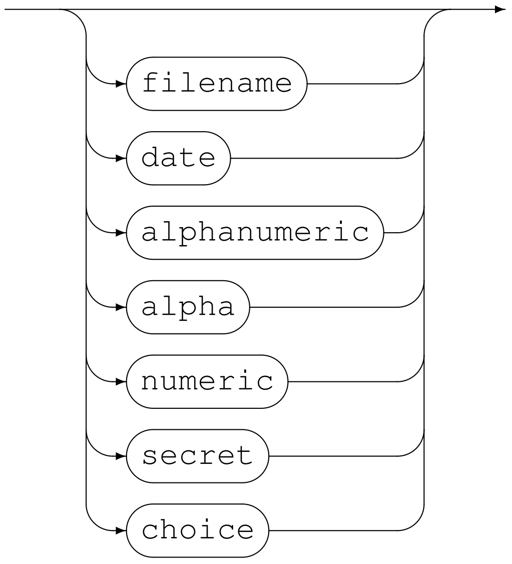
OptionsAttribute



OptionList



Option



The `secret` option means that the value is never held by the library in clear text. The value is encrypted immediately it is set from the user interface and only decrypted when the parameter value is retrieved. A secret parameter is also held in the log file as an encrypted value.

Some of the characteristics define a type for the parameter and as such are mutually exclusive of each other. The type of the parameter may be used by the user interface to present a specific dialogue to the user when updating the parameter; for example a standard Windows file-open dialogue may be presented for a parameter with a type of 'filename'. The type of the parameter also adds a further internal constraint on the value of the parameter; In other words the value supplied by the user must match BOTH the parameter constraint (refer to section 2.2.14 on page 11) and the type's internal constraint. The type names and the constraint each one imposes are as follows:

- date - Format CCYYMMDD

  `"^\(19\|20\)[0-9][0-9][0-1][0-9][0-3][0-9]$"`

- alphanumeric - Letters and Numbers

  `"^[a-zA-Z0-9]\+$"`

- alpha - Letters Only

  `"^[a-zA-Z]\+$"`

- numeric - Numbers Only

  `"^[0-9]\+$"`

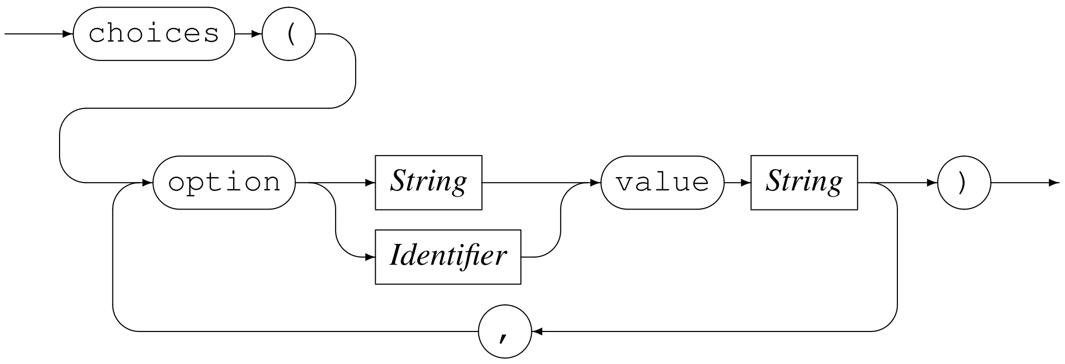
- filename - Operating System File Name

- Windows `"^[a-zA-Z0-9\:_.]\+$"`
- z/OS `"^[a-zA-Z0-9/_.()]\+$"`
- Linux/Unix `"^[a-zA-Z0-9/_.]\+$"`

• Choices

A parameter that defines a list of choices ignores any constraints imposed by the `constraint` keyword, as the only value that may be set for the parameter is one of the defined choices.

Choices



A choice may be either a `String` or an `Identifier` and the associated value must be a `String`.

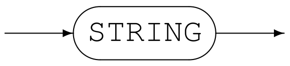
• Not Set or set to any other attribute - Allows Anything

`".*"`

2.2.17 *Value*

A value is a string concatenation. See section 2.2.18 on page 13 for information on writing long strings.

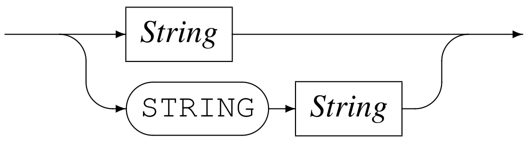
Value



2.2.18 *STRING*

A *STRING* is a concatenation of one or more *Strings*. This enables writing *Strings* that span multiple lines in the configuration file. For example a description can be written as 5 separate *Strings* over as many lines in the configuration. Once the configuration is loaded the strings are concatenated together to form one complete string.

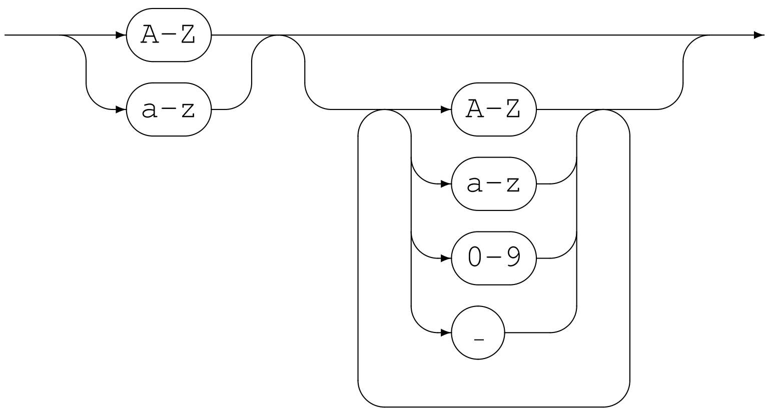
STRING



2.2.19 *Identifier*

An *Identifier* is a word that begins with an upper or lower case A through Z and is followed by any number of upper or lower case letters or the numerals 0 through nine or the underscore character.

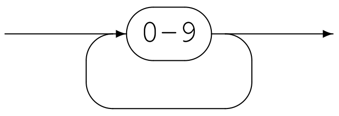
Identifier



2.2.20 *Number*

A *Number* is an integer comprising only of the numerals 0 through nine.

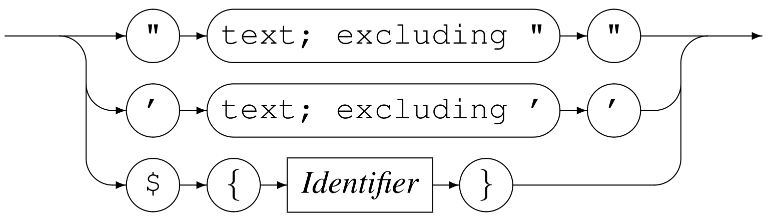
Number



2.2.21 *String*

A *String* is printable text enclosed by double or single quotes. The enclosing quote character may not appear in the text. A string may also be the result of resolving an environment variable reference. The format of an environment variable reference is shown below.

String



2.3 Configuration File Example

```
application SampleApplicationParameters;
-- This sample APD file defines the parameters used by an application.

-- $Author: hayward $
-- $Date: 2017/02/23 11:33:41 $
-- $Id: SampleApplicationParameters.apd,v 1.4 2017/02/23 11:33:41 hayward Exp $
-- $Name:  $
-- $Revision: 1.4 $
-- $State: Exp $
--
-- $Log: SampleApplicationParameters.apd,v $
-- Revision 1.4  2017/02/23 11:33:41  hayward
-- Allow options none to be specifically
-- stated - so the line does not have to
-- be removed.
--
-- Revision 1.3  2014/09/25 08:23:01  hayward
-- Change choice to be more informative.
--
-- Revision 1.2  2013/12/17 15:00:34  hayward
-- Correct parameters in choice and show more
-- detail in the test program when displaying
-- the choice parameter.
--
-- Revision 1.1  2013/12/17 14:08:11  hayward
-- Add CHOICE parameter type. This type does not
-- use the constraint attribute, but forces the
-- caller to choose one of multiple named choices
-- each of which relate to a different parameter
-- value for the calling application.
--

   title "Sample Application Parameters Configuration file";
   description "This configuration file is an example of how application "
     "parameters are defined. It shows an example of most of the different "
     "parameter types."
     ;

-- Set any environment variables that may be used in the configuration.
   set LOGHOME = ${HOME} "/logs";
   set TODAY = ${DATE_YYYYMMDD};

-- Define the directory in which the log of changes to parameter values, if any,
-- are stored.
   store ${LOGHOME};

-- The interface defines the shared object or DLL program that will interact
-- with the user to ensure that all parameters have a value. The value
-- for interface is string in quotes naming the DLL program or the word default
-- in which case the UI is the command line UI.

   interface default;
   entry none;

   parameter RUNDATE
      title "Run Date";
      default ${TODAY};
      options date;
      description "Set the run date to produce a report with the chosen "
         "effective date. The format of the supplied date must be CCYYMMDD. "
         "If it is not supplied by the user then todays date is used. "
         ;
      constraint "^\(19\|20\)[0-9][0-9][0-1][0-9][0-3][0-9]$";
   end

   parameter SYSID
      title "System Name";
      default ${SYSTEM_SID};
      options none;
      description "The current name of the system this process is running on.";
      constraint "^.*$";
   end

   parameter INFILE
      title "Input File";
      default NULL;                 -- Value required to be entered.
      options filename;
      description "Name of the raw input file to process in order to produce "
         "the report. A value must be supplied; either from the previous value "
         "or a new one entered at run time. "
         ;
      constraint "^[^ ]\+$"; --- at least one character, no spaces.
   end

   parameter OutputFileName
      title "The File to to be Written out";
      default NULL;
      options filename;
      description
         "This is the name of the output file that will be created "
         "as the result of a successful execution of this application."
         ;
      constraint "^[^ ]\+$"; --- at least one character, no spaces.
   end

   parameter FTPPassword
      title "Password for the FTP upload of the Output File";
      default NULL;
      options secret;  -- Value must not be shown by the UI.
      description
         "This is the password required when uploading the output file "
         "created by this process."
         ;
      constraint "^[^ ]\+$"; --- at least one character, no spaces.
   end

   parameter OutputFileRecioOpenString
      title "This parameter defines the Recio open string of the output file. "
         "The output file name is substituted for the %s, which is required "
         "in all of the choices.";
      default NULL;
      options choice;  -- Value must be one of the choices given
      description
         "Choose an output open string to match that required."
         ;
      constraint ".*";        -- ignored for a choice parameter.
      choices
         (
            option "Binary file" value "binary(%s,mode=rb,recfm=v)",
            option "Text File" value "text(%s,mode=r)",
            option NULLFILE value "null(%s)"
         );
   end

end.
```

3 API for an Application

3.1 Introduction

This interface is used by an application that requires an `Application Parameter` configuration file to be loaded in order to resolve run time parameter values. The code definition for this interface is held in the header file `applparms.h` shown in appendix A on page 41

The following sections describe the function calls that are available to an application that wishes to invoke the `applparms` library.

3.2 `applparms_open`

3.2.1 Synopsis

```
applparms_conf_t *applparms_open(char *filename, unsigned long flags);
```

3.2.2 Description

This is the first call an application must make to the `applparms` library and instantiates the `applparms` environment for the specified configuration file. If successful, it will return zero and all the parameters defined in the configuration file will have a value. If an error occurred during the open or the during the user interface parameter update phase the user selected to abort the process then `NULL` is returned and an error message can be retrieved by calling `applparms_error()`. The handle returned by this call must be passed back to the library on each subsequent call. The only exception is if the open fails then `NULL` can be used when calling `applparms_error()`.

If `flags` is set to use offline mode (see section A on page 41) then the user interface program is not called and the `applparms_open()` will return an error message if not all the parameters have a value after loading the configuration file and applying the history log.

3.3 `applparms_close`

3.3.1 Synopsis

```
int applparms_close(applparms_conf_t *applparms);
```

3.3.2 Description

This call will release all resources held by the `applparms` library, including the `applparms` handle returned on the open. No further calls to the `applparms` library are valid after this call.

3.4 `applparms_error`

3.4.1 Synopsis

```
char *applparms_error(applparms_conf_t *applparms);
```

3.4.2 Description

This function returns a pointer to the `NULL` terminated string that describes the error that occurred during an unsuccessful call to the `applparms` library.

3.5 `applparms_getparm_name`

3.5.1 Synopsis

```
char *applparms_getparm_name(applparms_conf_t *applparms, char *name);
```

3.5.2 Description

This function returns a pointer to a `NULL` terminated string that represents the value of the named parameter. If an error occurs then `NULL` is returned and an error message can be obtained by calling `applparms_error()`.

Be aware that if a parameter has the `secret` attribute set then it should not be displayed in clear text within the application; See the flag `APPLPARM_PARM_SECRET` in section A on page 41.

3.6 `applparms_getparm_parm`

3.6.1 Synopsis

```
char *applparms_getparm_parm(applparms_conf_t *applparms
      applparms_parm_t *parm);
```

3.6.2 Description

This function is similar to `applparms_getparm_name` except that the address of the parameter is passed instead of the name.

Be aware that if a parameter has the `secret` attribute set then it should not be displayed in clear text within the application; See the flag `APPLPARM_PARM_SECRET` in section A on page 41.

3.7 `applparms_enum_start`

3.7.1 Synopsis

```
void applparms_enum_start(applparms_conf_t *applparms);
```

3.7.2 Description

This function may be used by the calling application to set the current position of an enumeration of all parameters in the configuration file to the first defined parameter. This function, along with `application_enum_next` enables an application to enumerate through all the parameters defined in the configuration file. See section 3.10 on page 21 for an example.

3.8 `applparms_enum_next`

3.8.1 Synopsis

```
void applparms_enum_next(applparms_conf_t *applparms);
```

3.8.2 Description

This function may be used by the calling application to set the current position of an enumeration of all parameters in the configuration file to the next defined parameter. If this function or `applparms_enum_start` has not been called before then this function sets the current position to the first defined parameter. This function, along with `application_enum_next` enables an application to enumerate through all the parameters defined in the configuration file. See section 3.10 on page 21 for an example.

3.9 `applparms_get_current_name`

3.9.1 Synopsis

```
char *applparms_get_current_name(applparms_conf_t *applparms);
```

3.9.2 Description

This function returns the name of the parameter that is set as the current parameter by either of the two functions `applparms_enum_start` or `applparms_enum_next`. See section 3.10 on page 21 for an example. If an error occurs NULL is returned and the associated error message can be retrieved by calling `applparms_error()`.

3.10 `applparms_get_current_value`

3.10.1 Synopsis

```
char *applparms_get_current_value(applparms_conf_t *applparms);
```

3.10.2 Description

This function returns the value of the parameter that is set as the current parameter by either of the two functions `applparms_enum_start` or `applparms_enum_next`. If an error occurs NULL is returned and the associated error message can be retrieved by calling `applparms_error()`.

3.10.3 Example

Assume that an application has successfully opened and processed its application parameters configuration file and the pointer variable `applparms` is valid. An application can then perform the following to iterate and print the name and value of each parameter in the configuration:

```
char *name;
char *value;

fprintf(stderr,"%s\n",
   "Displaying all parameters via enum_start(next)");
for (applparms_enum_start(applparms);
      (name = applparms_get_current_name(applparms));
      applparms_enum_next(applparms))
   {
   value = applparms_get_current_value(applparms);
   if (!value)
      {
      fprintf(stderr,
         "testappl: Error getting value via enum_start: %s\n",
         applparms_error(NULL));
      break;
      }
   fprintf(stderr,"testappl: ENUM API %s=%s\n",name,value);
   }
fprintf(stderr,"%s\n","Displaying all parameters: END");
```

4 API for the User Interface

4.1 Introduction

The last action of the applparms library open function is to call the specified user interface to complete the update of the parameter values. The header file apui.h shown in appendix B on page 48 describes this interface and a pointer to the structure apui_t is passed to the user interface. The user interface program also needs to include the header applparms.h shown in appendix A on page 41.

4.2 Input and Usage

The user interface is passed a pointer to the applparms structure and a number of callback function pointers. The applparms structure pointer gives direct access the the in memory applparms configuration, but should be used only for iterating through the the following link lists:

- The parameter link list chained off the applparms structure and member parm_head.
- The mark points link list chained off the applparms structure and member mark_head.
- The parameter value history link list chained off the parameter structure and member history_head.

All other access (for example setting a parameter or mark) should be done through the supplied call back functions.

4.2.1 Iterating through Parameters

This example prints the parameter names in the exact order they were defined in the configuration file.

```
applparms_conf_t *applparms;
applparms_parm_t *parm;

applparms = apui->applparms;
for (parm = applparms->parm_head; parm; parm = parm->next)
   printf(" %s\n",parm->name);
```

4.2.2 Iterating through Mark Points

This code snippet prints out all the mark points as they are found chronologically in the log file.

```
applparms_conf_t *applparms;
applparms_mark_t *mark;

applparms = apui->applparms;
for (mark = applparms->mark_head; mark; mark = mark->next)
   printf(" %s %s %s\n",mark->timestamp,mark->name, mark->description);
```

4.2.3 Iterating through Parameter Value History

This code snippet prints out the parameter history values for each defined parameter.

```
applparms_conf_t *applparms;
applparms_history_t *history;

applparms = apui->applparms;
for (parm = applparms->parm_head; parm; parm = parm->next)
   {
   printf(" Parameter history for %s\n",parm->name);
   for (history = parm->history_head; history; history = history->next)
      printf(" %s %s %s\n",history->timestamp,
            history->name, history->description);
   }
```

4.3 Returning Control

The user interface should return an integer to the `applparms` library. If all parameters have a value then control can be returned and zero passed back to the library in order to indicate that the caller of the user interface library can continue with its processing. The value `-1` should be returned under the following conditions:

- An error occurs.
- There are one or more parameters without a value and the user elects to abort the parameter update process. See the function `apui_can_exit` in sub section 4.13 on page 28.

In both instances a detailed message, no longer than `APPLPARM_MAX_ERROR_LENGTH` bytes long should be copied into the `applparms` structure member `last_error` as in this example:

```
strncpy(apui->applparms->last_error,
      "User aborted Parameter updates",APPLPARM_MAX_ERROR_LENGTH);
return -1;
```

4.4 User Interface Call Back Functions

The following sections describe the call back function calls that are available to the user interface invoked by the `applparms` library.

4.5 `apui_setparm_name`

4.5.1 Synopsis

```
typedef int(*apui_setparm_name_t)(applparms_conf_t *applparms,
      char *parameter, char *value);
```

4.5.2 Description

This will set the named parameter to the value given. If successful then zero is returned, otherwise `-1` is returned and the associated error message can be obtained by calling the call back function `apui_error`.

4.6 `apui_setparm_parm`

4.6.1 Synopsis

```
typedef int(*apui_setparm_parm_t)(applparms_conf_t *applparms,
      applparms_parm_t *parameter, char *value);
```

4.6.2 Description

This call is similar to `apui_setparm_name` except that the address of the parameter is passed as input rather than its name.

4.7 `apui_setparm_default`

4.7.1 Synopsis

```
typedef int (*apui_setparm_default_t)(applparms_conf_t *applparms,
      char *name);
```

4.7.2 Description

This call will set the named parameter to its default value. If the command succeeds then zero is returned, otherwise `-1` is returned and an error message can be obtained by calling the call back function `apui_error`.

4.8 apui_getparm_name

4.8.1 Synopsis

```
typedef char *(*apui_getparm_name_t)(applparms_conf_t *applparms,
      char *name);
```

4.8.2 Description

This will return a pointer to the `NULL` terminated string that represents the value of the named parameter. If successful then zero is returned, otherwise `NULL` is returned and the associated error message can be obtained by calling the call back function `apui_error`.

Be aware that if a parameter has the `secret` attribute set then it should not be displayed in clear text within the user interface; See the flag `APPLPARM_PARM_SECRET` in section A on page 41.

4.9 apui_getparm_parm

4.9.1 Synopsis

```
typedef char *(*apui_getparm_parm_t)(applparms_conf_t *applparms,
      applparms_parm_t *parm);
```

4.9.2 Description

This call is similar to `apui_getparm_name` except that the address of the parameter is passed as input rather than its name.

Be aware that if a parameter has the `secret` attribute set then it should not be displayed in clear text within the user interface; See the flag `APPLPARM_PARM_SECRET` in section A on page 41.

4.10 apui_setmark

4.10.1 Synopsis

```
typedef int (*apui_setmark_t)(applparms_conf_t *applparm, char *mark,
      char *description);
```

4.10.2 Description

This call will set a mark point in the log file. The mark will have the current date and time and the description is set from the input parameter description. If successful then zero is returned, otherwise `-1` is returned and the associated error message can be obtained by calling the call back function `apui_error`.

4.11 apui_applylog

4.11.1 Synopsis

```
typedef int (*apui_applylog_t)(applparms_conf_t *applparms, char *mark,
      char *timestamp);
```

4.11.2 Description

This call will reset all parameter values to the value it was at the time the given mark point or timestamp as found in the log file. Normally only one of mark and timestamp should be set, the other being set to `NULL`. If both mark and timestamp are `NULL` then the log is applied to the end. If both mark and timestamp are supplied then the first one to be reached causes the apply process to complete.

Once the parameter values have been reset the state is recorded in the log file so that on restart all the parameters will have these reset values.

If successful then zero is returned, otherwise `-1` is returned and the associated error message can be obtained by calling the call back function `apui_error`.

4.12 apui_error

4.12.1 Synopsis

```
typedef char *(*apui_error_t)(applparms_conf_t *applparms);
```

4.12.2 Description

This function will return a pointer to the `NULL` terminated string that describes the error encountered during the last call back function called.

4.13 `apui_can_exit`

4.13.1 Synopsis

```
typedef int (*apui_can_exit_t)(applparms_conf_t *applparms);
```

4.13.2 Description

This call returns the number of parameters that do not have a value. If the return value is zero then the user interface may exit normally, otherwise a formatted error message can be obtained by calling the call back function `apui_error` and the user interface should not exit normally. The error message will name as many of the parameters that do not have a value as possible; if there are too many to fit in the message buffer the error message will end with three dots ('...').

5 The User Interfaces

5.1 Introduction

In order to complete the setting of all the parameter values the `applparms` library will call a defined user interface once the application's parameter configuration file has been loaded and their values set to the current state.

5.2 Definition

The user interface is defined in the parameter configuration file by the *InterfaceStatement*; see section 2.2.8 on page 9. This defines the program and entry point names of the program. If the default option is taken then the entry point is ignored as this interface is built into the `applparms` library.

5.3 Processing

The current state of the parameter values is initially that defined in the configuration file state and parameters that have no default value will therefore have no value. Once a value is set for a parameter the `applparms` library will remember it and it will be restored the next time the application is run.

The user interface is responsible for presenting the user at the terminal with an interface by which they can:

- View the value and attributes of one or more of the parameters.
- Set the value of an uninitialised parameter.
- Change the value of a parameter.
- Set a mark point. Mark points define the state of the parameter values at a specific time.
- View all the mark points.
- Roll back all parameter values to a previous mark point or timestamp.

5.4 Available User Interfaces

The following user interfaces are currently available and are documented in the sections following this one. They are namely:

1. The tabs graphical user interface. This may only be used on Windows. This allows applications to see all the parameters on one dialogue and to easily update the values or set parameters to a different state. See section 7 on page 37 for more information.
2. The list graphical user interface. This may only be used on Windows. This interface offers the same functionality as the tabs interface except that all the parameters are shown as a list. This interface can process many more parameters than the tabs one and should be used if the number of parameters is so large that the tabs are not easily readable. See section 8 on page 39 for more information.
3. The default interface. This interface may not be used from Windows GUI application as they do not normally have a command line available. It offers a command line interface in order to manipulate the parameter values. See section 6 on page 31 for more information.

6 The Default User Interface

6.1 Introduction

The default `applparms` user interface is a command line interface that allows the user to update and view parameter values, view parameter properties and roll back the parameter values to a stored mark point or timestamp.

When invoked the interface displays the start up message and the command line prompt. The prompt serves as a reminder that all commands entered are for the running application and not to the shell from which the application started from. This interface is available on any platform where the application is started from or is connected to a `stdin` and `stdout` console such as a DOS command window or a Unix shell.

6.2 Initial message and prompt

The initial message and prompt is as follows:

```
Application Parameter Default Command Interface.
  Use the Application Parameters Command interface to
  update and review parameter values or apply values
  from the log file. Use "help" for help on commands.
apui>
```

6.3 Examples of commands and responses

6.3.1 Showing the header information

This example shows how to list the configuration file header information.

```
apui>show header
show header
 Name:        GenerateDailySettlementFile
 Title:       Generate Daily Test Settlement File
 Description: This parameter member is used to define the parameters and their
              attributes used when running the eresia script that generates the
              daily settlement file ready for uploading to MVS for the nightly
              batch.
 Store:       .
 Interface:   Not Set
 Entry point: none
 Log file:    ./hayward_GenerateDailySettlementFile.apl
 Parm head@:  0x0806f558
 Parm tail@:  0x08077080
 Mark head@:     (nil)
 Mark tail@:     (nil)
 Log Start:   2010-01-20T12:22:55
 Log End:     2010-01-20T12:22:55
apui>
```

6.3.2 Showing parameter detail

The show command with a parameter name will show the attributes and value of the parameter. The command without a parameter name will show information for all the parameters.

```
apui>show RUNDATE
show RUNDATE
 Parameter:       RUNDATE
   Title:         Run Date
   Description:   Set the run date to produce a report with the chosen
                  effective date. The format of the supplied date must be
                  CCYYMMDD. If it is not supplied by the user then todays date
                  is used.
   Constraint:    "^\(19\|20\)[0-9][0-9][0-1][0-9][0-3][0-9]$"
   Default:       "20100120"
   Value:         "20100120"
   History head@: 0x08074810
   History tail@: 0x08074810
   Flags:         0x40000000
   Next:          SYSID
apui>
```

6.4 Syntax

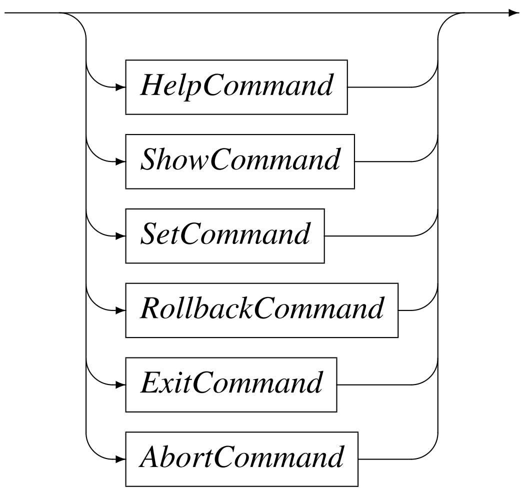
6.4.1 Introduction

The following sections show the syntax of the commands available in the default command interface and give some explanation of what they do. While in the interface the help command will give syntax details of the commands.

6.4.2 *Command*

A command comprises one of the sub commands as follows:

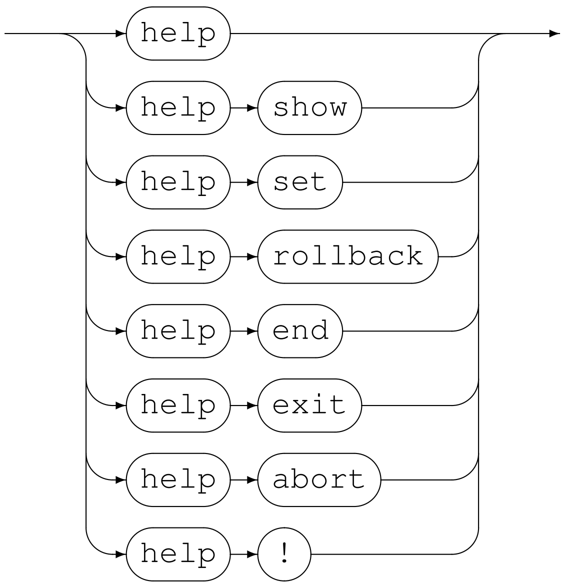
Command



6.4.3 *HelpCommand*

The help command gives detailed help on other commands.

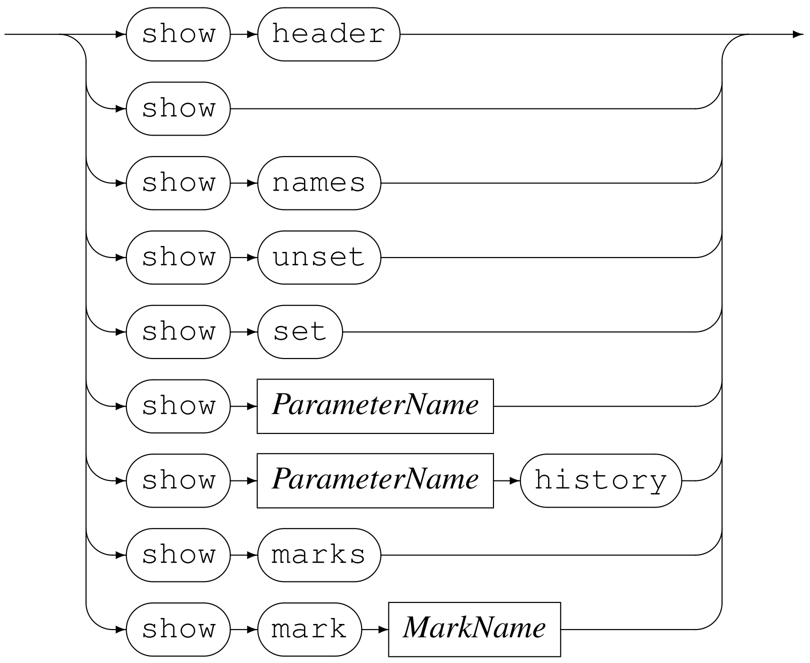
HelpCommand



6.4.4 *ShowCommand*

The show command can show the `applparms` header information, all parameters, all parameter names, all unset parameters names, all set parameter names, a specific parameter, all mark points or a specific mark point.

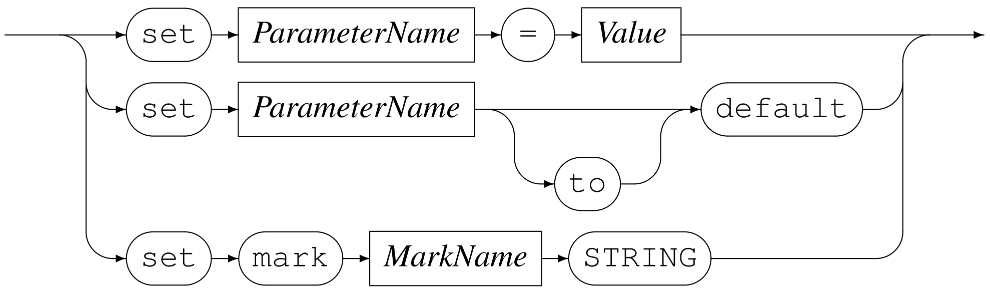
ShowCommand



6.4.5 *SetCommand*

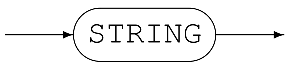
The set command is used to set a parameter to a value and record that change in the log; and to set a mark point in the log.

SetCommand



6.4.6 *Value*

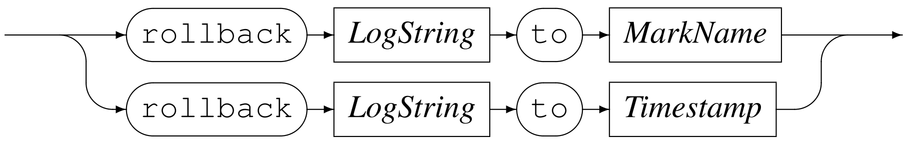
Value



6.4.7 *RollbackCommand*

The rollback command is used to set all parameters values back to those at a particular mark or time.

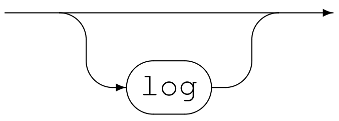
RollbackCommand



Carefully check for any error messages from the rollback command. If an error does occur then no assumption should be made about the state of the values of the parameters as some may have been reset and some not.

6.4.8 *LogString*

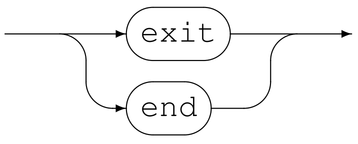
LogString



6.4.9 *ExitCommand*

The exit command will exit the update process if and only if all parameters have a value. If they do not then this command will show which parameters are still unset.

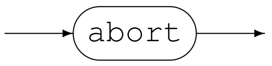
ExitCommand



6.4.10 *AbortCommand*

If the user wishes to abort the update process, possibly because it is impossible to give a parameter a value, then this command will exit the update process but return an error condition and message to the calling application.

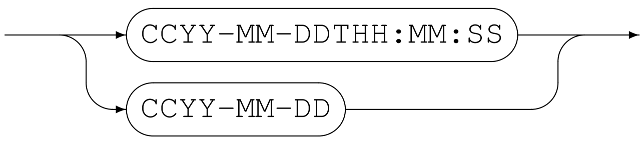
AbortCommand



6.4.11 *Timestamp*

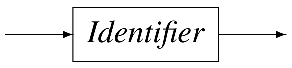
A timestamp that does not include the time will have the time portion defaulted to `23:59:59`.

Timestamp



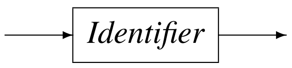
6.4.12 *ParameterName*

ParameterName



6.4.13 *MarkName*

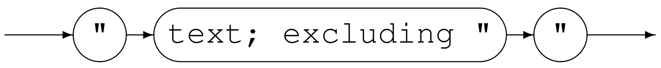
MarkName



6.4.14 *STRING*

A *String* is printable text enclosed by double quotes. The enclosing quote character may not appear in the text.

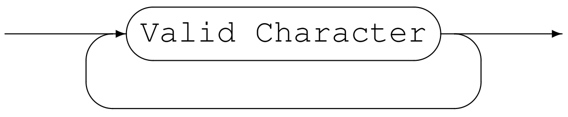
STRING



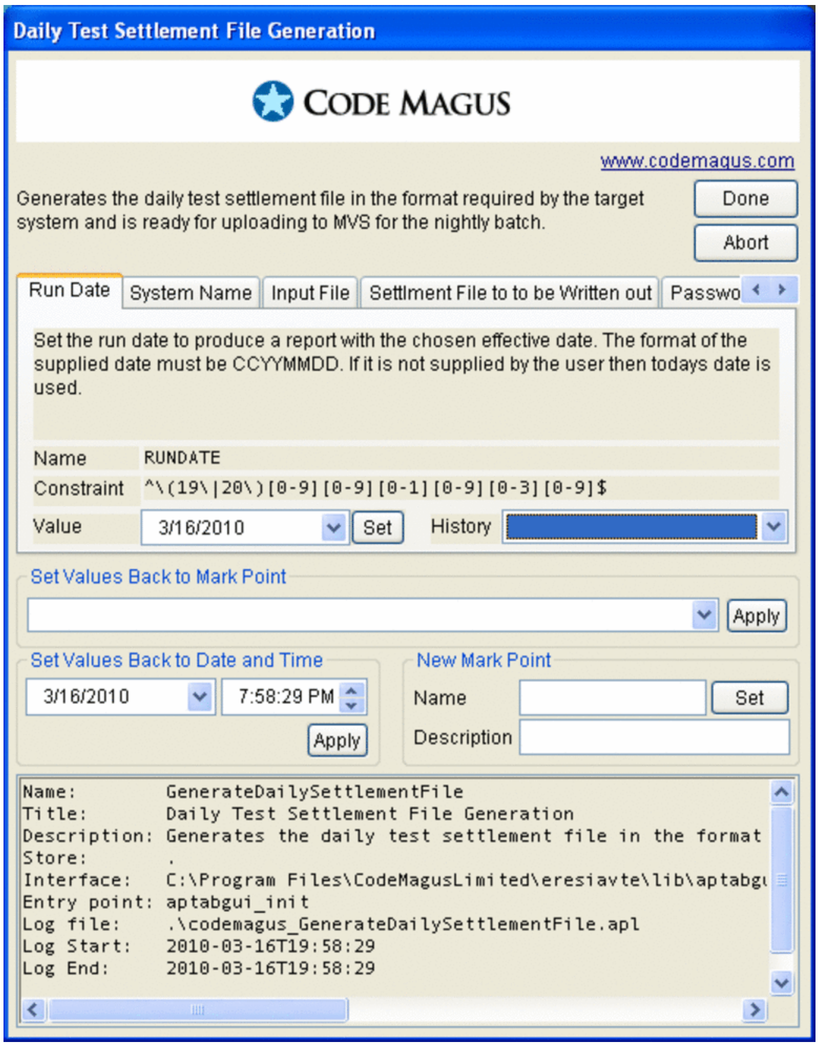
6.4.15 *Identifier*

An *Identifier* is a non-keyword word made up of valid characters. A valid character is a printable character excluding the double quote, newline or space character.

Identifier



7 The Tabs Graphical User Interface


Daily Test Settlement File Generation
CODE MAGUS
www.codemagus.com
Done
Abort
Run Date
System Name
Input File
Settlment File to to be Written out
Passwo
Set
History
Set Values Back to Mark Point
Apply
Set Values Back to Date and Time
Apply
New Mark Point
Set

Generates the daily test settlement file in the format required by the target system and is ready for uploading to MVS for the nightly batch.

Set the run date to produce a report with the chosen effective date. The format of the supplied date must be CCYYMMDD. If it is not supplied by the user then todays date is used.

Name	RUNDATE
Constraint	^\(19|20\)[0-9][0-9][0-1][0-9][0-3][0-9]$
Value	3/16/2010

Set Values Back to Date and Time: 3/16/2010 7:58:29 PM

New Mark Point: Name, Description

```
Name:        GenerateDailySettlementFile
Title:       Daily Test Settlement File Generation
Description: Generates the daily test settlement file in the format
Store:       .
Interface:   C:\Program Files\CodeMagusLimited\eresiavte\lib\aptabgu
Entry point: aptabgui_init
Log file:    .\codemagus_GenerateDailySettlementFile.apl
Log Start:   2010-03-16T19:58:29
Log End:     2010-03-16T19:58:29
```

7.1 Introduction

The tabs GUI interface is a Windows based dialogue that presents a single dialogue window that describes the configuration and using a tab form allows the user to set or update any parameter value as required.

7.2 Dialogue layout

The dialogue is divided into four sections as described below.

7.2.1 Description Section

At the top of the dialogue is the description section. It shows the description of the application parameters in effect and on the right are two buttons:

Done Clicking on this button will exit the dialogue if and only if all the parameters have a value. If not all the parameters have a value then the error message will give an indication of which parameters do not have a value.

Abort Clicking this button will immediately end the parameter update process whether or not all of the parameters have a value. It will return an error (and message) to the calling application that indicates that the user selected to abort the parameter update process.

7.2.2 Parameter Tabs Section

The next section is the parameter section. It shows all the parameters where each parameter is represented by a tab. By selecting a tab the user can view the attributes and value and set or change the value of the parameter. If there are more tabs than can fit in the dialogue the last tab shows three dots and the user can scroll the tab section left or right.

7.2.3 Mark Section

Below the tabs section is the mark section. This section has entry fields that:

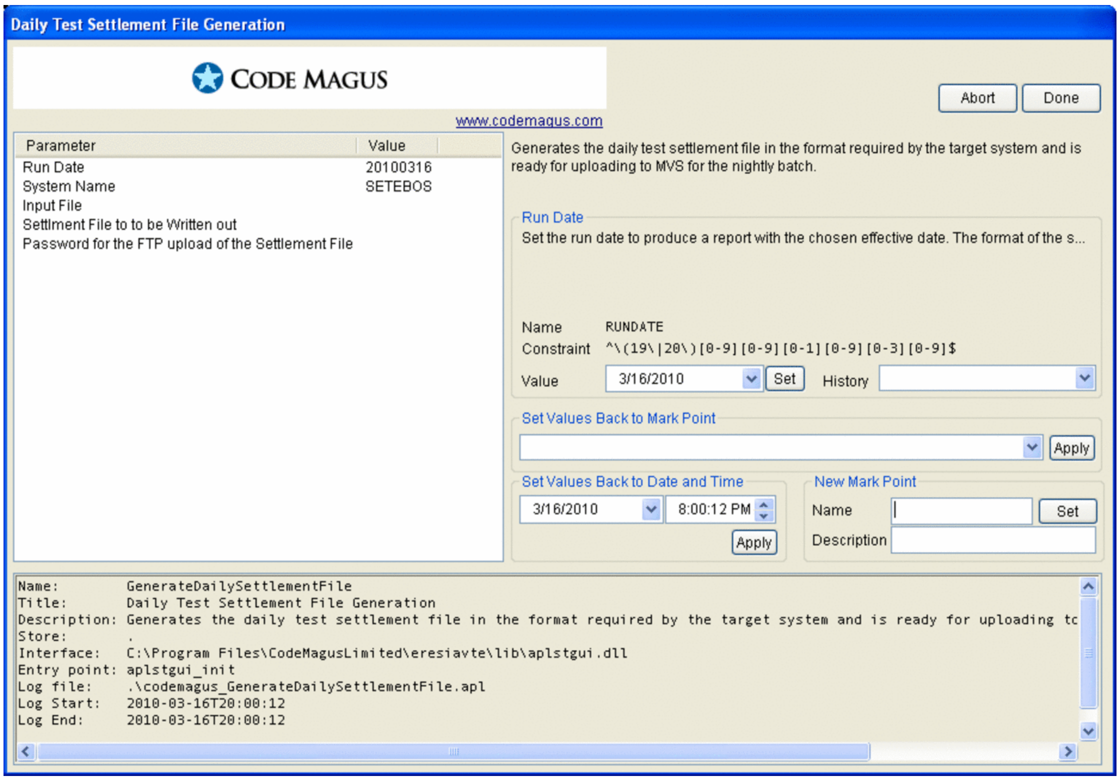
- show all the mark points that have been set.
- allow new mark points to be set.
- allow the user to roll back the state of the parameter values to a previously set mark point or timestamp.

Be aware that if an error occurs during a rollback operation the state of the values of the parameters is undefined as some may or may not have been reset.

7.2.4 Configuration File Header Section

Below the mark section is the header section that displays all the header properties of the application configuration file.

8 The List Graphical User Interface


Daily Test Settlement File Generation
CODE MAGUS
www.codemagus.com
Abort
Done
Set
History
Set Values Back to Mark Point
Apply
Set Values Back to Date and Time
New Mark Point
Set
Apply

Parameter	Value
Run Date	20100316
System Name	SETEBOS
Input File	
Settlment File to to be Written out	
Password for the FTP upload of the Settlement File	

Generates the daily test settlement file in the format required by the target system and is ready for uploading to MVS for the nightly batch.

Run Date

Set the run date to produce a report with the chosen effective date. The format of the s...

Name RUNDATE
Constraint ^\(19\|20\)[0-9][0-9][0-1][0-9][0-3][0-9]$
Value 3/16/2010

Set Values Back to Date and Time: 3/16/2010 8:00:12 PM

New Mark Point: Name, Description

```
Name:        GenerateDailySettlementFile
Title:       Daily Test Settlement File Generation
Description: Generates the daily test settlement file in the format required by the target system and is ready for uploading tc
Store:       .
Interface:   C:\Program Files\CodeMagusLimited\eresiavte\lib\aplstgui.dll
Entry point: aplstgui_init
Log file:    .\codemagus_GenerateDailySettlementFile.apl
Log Start:   2010-03-16T20:00:12
Log End:     2010-03-16T20:00:12
```

8.1 Introduction

The list GUI interface is a Windows based dialogue that presents a single dialogue window that describes the configuration and using a list form allows the user to set or update any parameter value as required.

8.2 Dialogue layout

The dialogue is divided into four sections as described below.

8.2.1 Description Section

At the top right of the dialogue is the description section. It shows the description of the application parameters in effect and above it are two buttons:

Done Clicking on this button will exit the dialogue if and only if all the parameters have a value. If not all the parameters have a value then the error message will give an indication of which parameters do not have a value.

Abort Clicking this button will immediately end the parameter update process whether or not all of the parameters have a value. It will return an error (and message) to the calling application that indicates that the user selected to abort the parameter update process.

8.2.2 Parameter List and details Section

The parameter list section is on the top left of the dialogue and the details section is to the right of this below the description section. It shows all the parameters where each parameter is represented by a line in the list. By selecting a line the user can view the attributes and value and set or change the value of the parameter in the details section. If there are more lines than can fit in the dialogue a scroll bar on the right of the list section allows the entries to be scrolled up or down.

8.2.3 Mark Section

Below the parameter details section is the mark section. This section has entry fields that:

- show all the mark points that have been set.
- allow new mark points to be set.
- allow the user to roll back the state of the parameter values to a previously set mark point or timestamp.

Be aware that if an error occurs during a rollback operation the state of the values of the parameters is undefined as some may or may not have been reset.

8.2.4 Configuration File Header Section

Across the bottom of the dialogue is the header section. This section displays all the header properties of the application parameters configuration file.

A applparms.h: Application Parameter Header File

This header file is used by an application that uses the applparms library to manage the set of parameters it needs.

```
#ifndef APPLPARMS_H
#define APPLPARMS_H
  /* File: applparms.h
   *
   * This file describes the interface to the Application Parameters Library
   * used in various tools and scripts. The library is responsible maintaining
   * a persistent store of personal choices made by the user; for making
   * sure that the values of the parameters entered by the user satisfy
   * certain constraints; and for allowing the user to see and to go back to
   * previous value sets.
   *
   * Author: Stephen R. Donaldson [stephen@codemagus.com].
   *
   * Copyright (c) 2009 by Code Magus Limited. All rights reserved.
   */

  /*
   * $Author: hayward $
   * $Date: 2013/12/17 14:08:11 $
   * $Id: applparms.h,v 1.13 2013/12/17 14:08:11 hayward Exp $
   * $Name:  $
   * $Revision: 1.13 $
   * $State: Exp $
   *
   * $Log: applparms.h,v $
   * Revision 1.13  2013/12/17 14:08:11  hayward
   * Add CHOICE parameter type. This type does not
   * use the constraint attribute, but forces the
   * caller to choose one of multiple named choices
   * each of which relate to a different parameter
   * value for the calling application.
   *
   * Revision 1.12  2011/01/28 17:48:44  hayward
   * Decryption uses a static buffer to hold the result.
   * This causes problems when more than one SECRET
   * parameter is declared as the second one then overwrites
   * the first and they both share the same address. This
   * change Creates another pointer in the parameter struct
   * to hold the decrypted value on a per parameter basis.
   * It is freed when a new decryption is performed or the
   * parameter is deleted.
   *
   * Revision 1.11  2010/02/17 14:07:19  hayward
   * Add support for parameters that have encrypted
   * values; in other words are secret. They are only
   * decrypted when the parameter is retreived for use.
   *
```

```
 * Revision 1.10  2010/02/03 17:37:11  hayward
 * Add functionality for offline processing.
 *
 * Revision 1.9  2010/01/11 13:53:27  hayward
 * Increase the size of the error message that can
 * be produced.
 *
 * Revision 1.8  2010/01/05 18:02:27  hayward
 * Changes from documentation review by SD.
 * This hilighted various changes to the
 * documentation and code:
 * 1. Changed PDF file name to include manual number.
 * 2. Changes to the Syntax and grammer of both the
 * config file and the default command interface.
 * 3. Added functionality to allow setting a parameter
 * back to its default value.
 * 4. Removed the log read function from the UI and
 * added a link list of marks and a link list of changes
 * to each parameter. The UI can use these to look at
 * history.
 * 5. Added a show history command for a parameter.
 *
 * Revision 1.7  2010/01/04 21:38:56  hayward
 * Fix comment typo.
 *
 * Revision 1.6  2009/12/29 09:10:54  hayward
 * Add Command UI DLL name and entry to config.
 * Added callback routine to check all parms are set.
 * Rollback now logs the current state to the log file.
 * command apply changed to rollback.
 *
 * Revision 1.5  2009/12/24 11:04:04  hayward
 * Change resulting from Memory checking and debugging.
 * Add hash of mark points.
 * Add internal constraint wrto parameter type.
 *
 * Revision 1.4  2009/12/21 15:22:16  hayward
 * Create a link list of the parmaters anchored
 * off the parm_head member in the config and
 * ordered exactly as they were defined.
 *
 * Revision 1.3  2009/12/16 12:28:50  hayward
 * Ratify the numerous headers into a caller
 * header (applparms.h) and a User interface
 * header (apui.h) and rename the ui programs.
 * This makes the program interfaces cleaner.
 * Also add setops for disallowing certain
 * function calls (set) when the context is
 * from the caller, but allow them in the
 * context of the UI. Add functionality to
 * the command parser and change how the
 * test program testappl works.
```

```
   *
   * Revision 1.2  2009/12/14 12:32:12  hayward
   * Changes for handling commands for
   * updating parameter values.
   *
   * Revision 1.1.1.1  2009/12/10 13:47:03  hayward
   * Import initial applparms to CVS.
   *
   */

#include <openssl/des.h>
#include <hashtab.h>
#include <regex.h>

  /*
   * Types and structures:
   */

typedef struct applparms_conf applparms_conf_t;
typedef struct applparms_parm applparms_parm_t;
typedef struct applparms_value_history applparms_value_history_t;
typedef struct applparms_mark_history applparms_mark_history_t;
typedef struct applparms_choice applparms_choice_t;

#define NO_VALUE    ((void *)1)           /* indicates no default value */

  /* codes used to restrict the commands that are allowed to execute */
typedef enum {
   APPLPARMS_SETPARM_NAME,
   APPLPARMS_SETPARM_PARM
   } applparms_functions_t;

  /* Structure applparms_conf_t is created as a result of processing an
   * application definition (or APD) file. The structure is only created
   * if the APD file can be successfully processed.
   */

struct applparms_conf
   {
   unsigned long flags;           /* options from open */
#define APPLPARMS_OPT_VERBOSE 0x80000000  /* produce verbose output when
                                             processing */
#define APPLPARMS_OPT_OFFLINE 0x40000000  /* process in non-interactive mode */
   char *name;                    /* application name from APD file */
   char *title;                   /* application title from APD file */
   char *description;             /* application description from APD file */
   char *config;                  /* APD file name */
   char *store;                   /* Users corresponding APL location */
   char *ui_interface;            /* Shared object / DLL to be used as UI */
   char *ui_entry;                /* Entry point of ui_interface */
   char *aplfile;                 /* Fully qualified APL file name */
   applparms_parm_t *parm_head;   /* first parameter defined */
   applparms_parm_t *parm_tail;  /* last parameter defined */
   applparms_parm_t *current_enum_parm; /* Current enumerated parm */
   hashtab_t *parms;             /* hash table of parms for direct access */
   applparms_mark_history_t *mark_head;  /* first mark history entry */
   applparms_mark_history_t *mark_tail;  /* last mark history entry */
   hashtab_t *marks;             /* hash table of log marks */
   char *last_error;             /* error accessible through applparms_error()*/
#define APPLPARM_MAX_ERROR_LENGTH 4000
   DES_cblock key[3];            /* DES keys */
   DES_cblock ivec;              /* Initialisation vector for DES */
   DES_key_schedule schedule[3]; /* DES Arch specific key schedules */
   char log_start_ts[19+1];      /* Log file start timestamp */
   char log_end_ts[19+1];        /* Log file end timestamp */
   };

  /* Structure applparms_parm_t describes one parameter and has all the
   * attributes taken from the definition of the parameter from the
   * APD file section describing that parameter.
   */

struct applparms_parm
   {
   applparms_parm_t *next; /* Next parm in the list */
   char *name;             /* name of the parameter as used by application */

   unsigned long flags;    /* options for processing the value of parm */
#define APPLPARM_PARM_FILENAME 0x80000000    /* value is local file name */
#define APPLPARM_PARM_DATE     0x40000000    /* value is a calender date */
#define APPLPARM_PARM_ALNUM    0x20000000    /* value is an alphanum string */
#define APPLPARM_PARM_ALPHA    0x10000000    /* value is an alpha string */
#define APPLPARM_PARM_NUMERIC  0x08000000    /* value is a numeric string */
#define APPLPARM_PARM_SECRET   0x04000000    /* value is a password */
#define APPLPARM_PARM_CHOICE   0x02000000    /* value is a choice of more than
                                                one item */

   char *title;            /* title of parameter as displayed to user */
   char *description;      /* long text description of parameter */
   char *constraint;       /* regular expression constraining parm value */
   char *default_value;    /* default value of parameter */
   char *value;            /* current value of parameter */
   char *value_decrypt;    /* Holder for decrypted value */
   applparms_value_history_t *history_head;  /* First Value History entry */
   applparms_value_history_t *history_tail;  /* last Value History entry */
   regex_t preg;           /* Compiled expression of constraint; This is over
                              and above the internal constraint that a user may
                              further impose on the value. */
   regex_t internal_preg;  /* Compiled expression of internal constraint; These
                              constraints hold true depending on the flag value
                              depicting the parameter type.*/
   applparms_choice_t *choices; /* Choices constraining a choice parameter */
   applparms_choice_t *choices_tail;   /* Tail of choices, for adding new */
#ifdef WIN32
```

```
#define APPLPARM_REGEX_FILENAME "^[a-zA-Z0-9\\:_.]\\+$"
#elif defined(__MVS__)
#define APPLPARM_REGEX_FILENAME "^[a-zA-Z0-9/_.()]\\+$"
#else
#define APPLPARM_REGEX_FILENAME "^[a-zA-Z0-9/_.]\\+$"
#endif
#define APPLPARM_REGEX_DATE "^\\(19\\|20\\)[0-9][0-9][0-1][0-9][0-3][0-9]$"
#define APPLPARM_REGEX_ALNUM "^[a-zA-Z0-9]\\+$"
#define APPLPARM_REGEX_ALPHA "^[a-zA-Z]\\+$"
#define APPLPARM_REGEX_NUMERIC "^[0-9]\\+$"
#define APPLPARM_REGEX_NONE ".*"
   };

  /* Structure applparms_history_t describes the history of one value of a
   * parameter.
   */

struct applparms_value_history
   {
   applparms_value_history_t *next; /* Next in chain */
   char *timestamp;                 /* Time value was set */
   char *value;                     /* Historical value */
   };

  /* Structure applparms_mark_t describes the history of one mark point.
   */

struct applparms_mark_history
   {
   applparms_mark_history_t *next;  /* Next in chain */
   char *timestamp;                 /* Time mark was set */
   char *name;                      /* mark name */
   char *description;               /* Description of Mark */
   };

  /* Structure applparms_choices describes a link list of the available choices
   * that a parameter may have.
   */
struct applparms_choice
   {
   applparms_choice_t *next;        /* Next entry */
   char *choice;                    /* Strings or identifiers presented to the
                                       user from which they make a choice. */
   char *value;                     /* value of choice - returned to a caller
                                       using applparms_getparm_parm() */
   };

  /* Function applparms_open() is called to parse the application definition
   * parameters file and returns a data structure corresponding to the contents
   * of the APD file to the caller. If a problem occurs during the processing
   * of the APD file and a data structure cannot be created then the function
   * returns NULL. The corresponding error message can be obtained by calling
   * the function applparms_error(). The data structure returned must be passed
   * back to all subsequent library calls.
   */

applparms_conf_t *applparms_open(char *filename, unsigned long flags);

  /* Function applparms_close() is called to free and release the resources
   * associated with an applparms_conf_t data structure and should be the last
   * call to use the data structure. The function returns 0 on success and -1
   * otherwise. If the function returns -1, an associated error message can be
   * obtained using the applparms_error() function. The data structure should
   * not be used after the applparms_close() call.
   */

int applparms_close(applparms_conf_t *applparms);

  /* Function applparms_error() returns the address of a NULL terminated error
   * message associated with the last error that has occurred. If the error was
   * created by the applparms_open() function call, or by the applparms_close()
   * function call then the applparms_error() function should be called with
   * NULL to retrieve the error message. If there is no prior error condition
   * then an empty string is returned.
   */

char *applparms_error(applparms_conf_t *applparms);

  /* Function applparms_setparm_name() is used to set the value of a parameter
   * in the data structure. The function is intended to be called by the library
   * or control which interfaces to the user. The function returns 0 if
   * successful, otherwise -1 is returned and an associated error message can
   * be obtained using the applparms_error() function.
   */

int applparms_setparm_name(applparms_conf_t *applparms, char *name,
      char *value);

  /* Function applparms_getparm_name() is used to get the value of a parameter
   * in the data structure. The function is intended to be called by or on
   * behalf of the application or script using the value of the variable. The
   * function is also intended to be called by the library or control
   * interfacing with the user in order to show the user the current value of
   * the parameter. The function returns the string point of the value if
   * successful and NULL if not in which case an associated message can be
   * obtained by calling the applparms_error() function.
   */

char *applparms_getparm_name(applparms_conf_t *applparms, char *name);

  /* Function applparms_setparm_parm() is similar to applparms_setparm_name()
   * except that the parm structure is passed to the function instead of the
   * parm name.
   */
```

```
int applparms_setparm_parm(applparms_conf_t *applparms, applparms_parm_t *parm,
      char *value);

  /* Function applparms_getparm_parm() is similar to applparms_getparm_name()
   * except that the parm structure is passed to the function instead of the
   * parm name.
   */

char *applparms_getparm_parm(applparms_conf_t *applparms,
      applparms_parm_t *parm);

  /* Function applparms_setmark() will write a timestamped entry to the log
   * file. This entry can be used as a marker to reset all values up to that
   * point. On success it returns zero, and -1 if not in which case an error
   * message can be obtained by calling applparms_error().
   */

int applparms_setmark(applparms_conf_t *applparms, char *name,
      char *description);

   /* Functions applparms_enum_start, applparms_enum_next,
    * applparms_get_current_name and applparms_get_current_value can be used to
    * enumerate the current set of application parameters in order to
    * retrieve the names and values of each parameter.
    */

void applparms_enum_start(applparms_conf_t *applparms);
void applparms_enum_next(applparms_conf_t *applparms);
char *applparms_get_current_name(applparms_conf_t *applparms);
char *applparms_get_current_value(applparms_conf_t *applparms);

#endif /* APPLPARMS_H */
```

B `apui.h`: Application User Interface Header File

This header is used by a developer that writes a new or maintains an old user interface. An interface program also requires the `applparms.h` header.

```
#ifndef APUI_H
#define APUI_H
  /* File: apui.h
   *
   * This header file describes the interface to the application parameter user
   * interface (UI) update processor. This allows a UI to interface with the
   * user in order to update all the defined Application Parameters. A default
   * command line UI is supplied when the application parameter library can not
   * locate or load an UI update processor.
   * Generally these interfaces are qritten as a DLL or shared object and
   * because they are passed the address of the applparms structure they have
   * complete access to all members in it; HOWEVER when retreiving or updating a
   * parameter value the DLL should use the callback functions supplied; the
   * reason being that there is more processing that may (or may not) occur
   * during the update of parameter and the maintenance of that is best left to
   * the library.
   *
   * Author: Stephen R. Donaldson [www.codemagus.com].
   *
   * Copyright (c) 2009 Code Magus Limited. All rights reserved.
   *
   */

  /*
   * $Author: hayward $
   * $Date: 2010/01/05 18:02:27 $
   * $Id: apui.h,v 1.5 2010/01/05 18:02:27 hayward Exp $
   * $Name:  $
   * $Revision: 1.5 $
   * $State: Exp $
   *
   * $Log: apui.h,v $
   * Revision 1.5  2010/01/05 18:02:27  hayward
   * Changes from documentation review by SD.
   * This hilighted various changes to the
   * documentation and code:
   * 1. Changed PDF file name to include manual number.
   * 2. Changes to the Syntax and grammer of both the
   * config file and the default command interface.
   * 3. Added functionality to allow setting a parameter
   * back to its default value.
   * 4. Removed the log read function from the UI and
   * added a link list of marks and a link list of changes
   * to each parameter. The UI can use these to look at
   * history.
   * 5. Added a show history command for a parameter.
   *
   * Revision 1.4  2010/01/04 21:38:24  hayward
   * Correct typedef function prototypes.
   *
   * Revision 1.3  2009/12/29 09:10:54  hayward
   * Add Command UI DLL name and entry to config.
   * Added callback routine to check all parms are set.
   * Rollback now logs the current state to the log file.
   * command apply changed to rollback.
   *
   * Revision 1.2  2009/12/16 12:28:50  hayward
   * Ratify the numerous headers into a caller
   * header (applparms.h) and a User interface
   * header (apui.h) and rename the ui programs.
   * This makes the program interfaces cleaner.
   * Also add setops for disallowing certain
   * function calls (set) when the context is
   * from the caller, but allow them in the
   * context of the UI. Add functionality to
   * the command parser and change how the
   * test program testappl works.
   *
   * Revision 1.1  2009/12/14 12:32:12  hayward
   * Changes for handling commands for
   * updating parameter values.
   *
   */

  /*
   * Structures:
   */
typedef struct applparms_apui applparms_apui_t;

  /*
   * Function Call backs:
   */

typedef int(*apui_setparm_name_t)(applparms_conf_t *applparms, char *parameter,
      char *value);
typedef int(*apui_setparm_parm_t)(applparms_conf_t *applparms,
      applparms_parm_t *parameter, char *value);
typedef int (*apui_setparm_default_t)(applparms_conf_t *applparms, char *name);
typedef char *(*apui_getparm_name_t)(applparms_conf_t *applparms,
      char *name);
typedef char *(*apui_getparm_parm_t)(applparms_conf_t *applparms,
      applparms_parm_t *parm);
typedef int (*apui_setmark_t)(applparms_conf_t *applparm, char *mark,
      char *description);
typedef int (*apui_applylog_t)(applparms_conf_t *applparms, char *mark,
      char *timestamp);
typedef char *(*apui_error_t)(applparms_conf_t *applparms);
typedef int (*apui_can_exit_t)(applparms_conf_t *applparms);
```

```
  /*
   * The applparms_apui user interface structure is passed to the UI handler
   * allowing it to make sure that all parameters are updated by the user.
   * Different UI programs operate on different platforms (GUI on Windows,
   * command line interface on DOS and Linux).
   */

struct applparms_apui
   {
   applparms_conf_t *applparms;
   apui_setparm_name_t apui_setparm_name;
   apui_setparm_parm_t apui_setparm_parm;
   apui_setparm_default_t apui_setparm_default;
   apui_getparm_name_t apui_getparm_name;
   apui_getparm_parm_t apui_getparm_parm;
   apui_setmark_t apui_setmark;
   apui_applylog_t apui_applylog;
   apui_error_t apui_error;
   apui_can_exit_t apui_can_exit;
   };

  /*
   * Exported Functions:
   */

int applparms_default_method(applparms_apui_t *apui);

#endif /* APUI_H */
```