BCI2000 - Functions Documentation

ModuleName	FunctionName	ClassName
UBCItime.cpp	~BCITIME()	BCITIME

Function

BCITIME::~BCITIME()

Purpose

the destructor for the BCITIME object

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

BCI2000 - Functions Documentation

ModuleName	FunctionName	ClassName
UBCItime.cpp	BCITIME()	BCITIME

Function

BCITIME::BCITIME()

Purpose

the constructor for the BCITIME object

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UBCItime.cpp	GetBCItime_ms()	BCITIME

Function

unsigned short BCITIME::GetBCItime_ms()

Purpose

gets the current time from Windows' high-performance timers

Parameters

Returns

an unsigned sixteen bit value for the current time in ms
(-> wrap-around occurs every 65536ms, or about 6.5 seconds)

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UBCItime.cpp	TimeDiff	BCITIME

Function

unsigned short BCITIME::TimeDiff

Purpose

calculates the difference between two times (i.e., time2-time1)
takes roll-over into account (in case time2 < time1)

Parameters

time1, time2 - two 16 bit integers

Returns

time2-time1, if time2-time1 >= 0 or time2-time1+65536, if time2-time1 < 0

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	~COREMESSAGE()	COREMESSAGE

Function

COREMESSAGE::~COREMESSAGE()

Purpose

the destructor for the COREMESSAGE object

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	COREMESSAGE()	COREMESSAGE

Function

COREMESSAGE::COREMESSAGE()

Purpose

the constructor for the COREMESSAGE object

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	GetBufPtr()	char *COREMESSAGE

Function

char *COREMESSAGE::GetBufPtr()

Purpose

Gets a pointer to the GetLength() bytes data in the coremessage

Parameters

N/A

Returns

pointer to the data

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	GetDescriptor()	BYTE COREMESSAGE

Function

BYTE COREMESSAGE::GetDescriptor()

Purpose

returns the descriptor for this coremessage

Parameters

N/A

Returns

descriptor

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	GetLength()	COREMESSAGE

Function

unsigned short COREMESSAGE::GetLength()

Purpose

Returns the length of this coremessage
This length specifies ONLY the length of the actual data,
i.e., it EXcludes the length of the header when it is being
transmitted. The length of a coremessage cannot exceed 65kB

Parameters

N/A

Returns

length of the core message

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	GetSuppDescriptor()	BYTE COREMESSAGE

Function

BYTE COREMESSAGE::GetSuppDescriptor()

Purpose

returns the supplemental descriptor for this coremessage

Parameters

N/A

Returns

supplemental descriptor

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	ParseMessage()	COREMESSAGE

Function

int COREMESSAGE::ParseMessage()
int COREMESSAGE::ParseMessage()

Purpose

After having received a coremessage, ParseMessage()
further processes the message. Depending on the type of the coremessage, it processes the state, parameter, or status,
and initializes the respective object (status, param, state) in
the coremessage object

Parameters

N/A

Returns

the type of the coremessage (as defined in UCoreMessage.h
or COREMSG_NONE, if the type is not recognized

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	ReceiveBufBytes(TCustomWinSock	COREMESSAGE

Function

int COREMESSAGE::ReceiveBufBytes(TCustomWinSocket *Socket, char *buf, int length)

Purpose

this procedure reads EXACTLY length bytes from the specified
(non-blocking) socket connection, i.e., it does not return
before

Parameters

Socket - the non-blocking socket
buf - the buffer that has been allocated to hold the data
length - the number of byes to read from the socket connection

Returns

length ... on success
0 ... on error

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	ReceiveBufBytes(TWinSocketStrea	COREMESSAGE

Function

int COREMESSAGE::ReceiveBufBytes(TWinSocketStream *stream, char *buf, int length)

Purpose

this procedure reads EXACTLY length bytes from the specified
Socket stream that has been created for a (blocking) socket
connection, i.e., it does not return before

Parameters

stream - the stream to the blocking socket
buf - the buffer that has been allocated to hold the data
length - the number of byes to read from the socket connection

Returns

length ... on success
0 ... on error

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	ReceiveCoreMessage(TCustomWin	COREMESSAGE

Function

int COREMESSAGE::ReceiveCoreMessage(TCustomWinSocket *Socket)

Purpose

given a handle to an open non-blocking Socket connection,
receives the content of a coremessage and sets the values
of the descriptor, the supplemental descriptor and the length
accordingly
use ParseMessage() to further process the coremessage

Parameters

Socket - the non-blocking socket

Returns

always ERRCORE_NOERR

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	ReceiveCoreMessage(TWinSocketS	COREMESSAGE

Function

int COREMESSAGE::ReceiveCoreMessage(TWinSocketStream *stream)

Purpose

given a handle to a Socket stream that has been created for
a given Socket connection,
receives the content of a coremessage and sets the values
of the descriptor, the supplemental descriptor and the length
accordingly
use ParseMessage() to further process the coremessage

Parameters

stream - an active stream

Returns

ERRCORE_NOERR success
ERRCORE_RECEIVEBUFBYTES on failure

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	SendBufBytes(TCustomWinSocket	COREMESSAGE

Function

int COREMESSAGE::SendBufBytes(TCustomWinSocket *Socket, char *buf, int length)

Purpose

this procedure sends EXACTLY length bytes to the specified
(non-blocking) socket connection, i.e., it does not return before

Parameters

Socket - the non-blocking socket
buf - the buffer that has been allocated to hold the data
length - the number of byes to read from the socket connection

Returns

<= 0 ... on error
>0 ... on success

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	SendCoreMessage(TCustomWinSoc	COREMESSAGE

Function

int COREMESSAGE::SendCoreMessage(TCustomWinSocket *Socket)

Purpose

given a handle to an open non-blocking Socket connection,
sends a coremessage, including the appropriate header
values for the descriptor, the supplemental descriptor and the length
have to be specified accordingly

Parameters

Socket - the non-blocking socket

Returns

ERRCORE_NOERR on success
ERRCORE_SENDBUFBYTES, if there was an error sending bytes out
ERRCORE_SOCKETNOTOPEN, if the socket connection was not open

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	SendCoreMessage(TWinSocketStre	COREMESSAGE

Function

int COREMESSAGE::SendCoreMessage(TWinSocketStream *stream)

Purpose

given a handle to an open Socket stream,
sends a coremessage, including the appropriate header
values for the descriptor, the supplemental descriptor and the length
have to be specified accordingly

Parameters

stream - the stream that has been created for a blocking socket connection

Returns

ERRCORE_NOERR on success
ERRCORE_WRITE on failure

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	SetDescriptor(BYTE newdescriptor)	COREMESSAGE

Function

void COREMESSAGE::SetDescriptor(BYTE newdescriptor)

Purpose

sets the descriptor for this coremessage

Parameters

newdescriptor - the new descriptor

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	SetLength(unsigned short newlength	COREMESSAGE

Function

void COREMESSAGE::SetLength(unsigned short newlength)

Purpose

Sets the length of this coremessage
This length specifies ONLY the length of the actual data,
i.e., it EXcludes the length of the header when it is being
transmitted. The length of a coremessage

Parameters

newlength - length of the core message

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UCoreMessage.cpp	SetSuppDescriptor(BYTE newsupp	COREMESSAGE

Function

void COREMESSAGE::SetSuppDescriptor(BYTE newsuppdescriptor)

Purpose

sets the supplemental descriptor for this coremessage

Parameters

newsuppdescriptor - the new supplemental descriptor

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParam.cpp	SetValue	PARAM

Function

void PARAM::SetValue(char *src, int idx)

Purpose

sets the idx'th value of the parameter

Parameters

src - char pointer to the value
idx - index of the value (0...GetNumValues()-1)

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	~PARAM()	PARAM

Function

PARAM::~PARAM()

Purpose

The destructor for the PARAM object
it deletes the list of values that this parameter holds

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	~PARAMLIST()	PARAMLIST

Function

PARAMLIST::~PARAMLIST()

Purpose

The destructor for the PARAMLIST object. It deletes both
all PARAM objects in the list and also the list object itself

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	AddParameter2List(char *paramstri	PARAMLIST

Function

void PARAMLIST::AddParameter2List(char *paramstring, int paramlen)

Purpose

adds a new parameter to the list of parameters
if a parameter with the same name already exists,
it updates the currently stored parameter with the provided values

Parameters

paramstring - ASCII string, as defined in project description,
defining this parameter

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	ClearParamList()	PARAMLIST

Function

void PARAMLIST::ClearParamList()

Purpose

clears and frees all entries in the parameter list

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	CloneParameter2List(PARAM *par	PARAMLIST

Function

void PARAMLIST::CloneParameter2List(PARAM *param)

Purpose

adds a new parameter to the list of parameters
The difference to MoveParameter2List() is that it actually
physically copies the param object (the specified param
object can then be freed elsewhere)
This function will UPDATE the values for a parameter in the list,
if one with the same name already exists

Parameters

param - pointer to an existing PARAM object

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	ConstructParameterLine()	PARAM

Function

int PARAM::ConstructParameterLine()

Purpose

Construct a parameter line, based upon the current values
in the PARAM object

Parameters

N/A

Returns

ERRPARAM_NOERR, or
ERRPARAM_INCONSISTENTNUMVAL, if the number of values in the
object is inconsistent with the actually present number of values

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	DeleteParam(char *name)	PARAMLIST

Function

void PARAMLIST::DeleteParam(char *name)

Purpose

deletes a parameter of a given name in the list of parameters
it also frees the memory for this particular parameter
it does not do anything, if the parameter does not exist

Parameters

name - name of the parameter

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	get_argument(int ptr, char *buf, cha	int PARAM

Function

int PARAM::get_argument(int ptr, char *buf, char *line, int maxlen)

Purpose

parses the parameter line that is being sent in the core
communication, or as stored in any BCI2000 .prm file
it returns the next token that is being delimited by either
a ' ' or '='

Parameters

ptr - index into the line of where to start
buf - destination buffer for the token
line - the whole line
maxlen - maximum length of the line

Returns

the index into the line where the returned token ends

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetListPtr()	TList *PARAM

Function

TList *PARAM::GetListPtr()

Purpose

Returns a pointer to this parameter's list of values

Parameters

N/A

Returns

pointer to the list

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetName()	char *PARAM

Function

char *PARAM::GetName()

Purpose

Returns a pointer to this parameter's name
Note: parameter names are case insensitive

Parameters

N/A

Returns

char pointer to the name

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetNumParameters()	PARAMLIST

Function

int PARAMLIST::GetNumParameters()

Purpose

Returns the number of parameters in the list

Parameters

N/A

Returns

the number of parameters

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetNumValues()	PARAM

Function

int PARAM::GetNumValues()

Purpose

Returns the number of values in this parameter
for most types of parameters, this will be one

Parameters

N/A

Returns

number of values in this parameter

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetParamLine()	char *PARAM

Function

char *PARAM::GetParamLine()

Purpose

Returns a parameter line in ASCII format
Tis parameter line is constructed, based upon the current
values in the PARAM object

Parameters

N/A

Returns

a pointer to the parameter line

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetParamPtr(char *name)	PARAM *PARAMLIST

Function

PARAM *PARAMLIST::GetParamPtr(char *name)

Purpose

given a parameter name, returns the pointer to a PARAM object

Parameters

name - name of the parameter

Returns

pointer to a PARAM object or
// NULL, if no parameter with this name exists in the list

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetParamPtr(int idx)	PARAM *PARAMLIST

Function

PARAM *PARAMLIST::GetParamPtr(int idx)

Purpose

given an index (0..GetListCount()-1), returns the pointer to a PARAM object

Parameters

idx - index of the parameter

Returns

pointer to a PARAM object or
NULL, if the specified index is out of range

Group	Responsible	EMailAddress
Albany	Gerv	schalk@Wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetSection()	char *PARAM

Function

char *PARAM::GetSection()

Purpose

Returns a pointer to this parameter's section name

Parameters

N/A

Returns

char pointer to the section name

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetType()	char *PARAM

Function

char *PARAM::GetType()

Purpose

Returns a pointer to this parameter's type

Parameters

N/A

Returns

char pointer to the type

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetValue()	char *PARAM

Function

char *PARAM::GetValue()

Purpose

Returns a pointer to the FIRST value of this parameter
(most parameters, except the *list parameter types, only
have one value anyways)

Parameters

N/A

Returns

char pointer to the value

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	GetValue(int idx)	char *PARAM

Function

char *PARAM::GetValue(int idx)

Purpose

Returns a pointer to the i'th value of this parameter
(most parameters, except the *list parameter types, only
have one value anyways)

Parameters

idx ... index of the value

Returns

char pointer to the value
if idx is out of bounds, it returns a pointer to the first value

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	MoveParameter2List(PARAM *new	PARAMLIST

Function

void PARAMLIST::MoveParameter2List(PARAM *newparam)

Purpose

adds a new parameter to the list of parameters
this function assumes that the specified parameter object
will not be freed anywhere else (but in the destructor of the
paramlist object). The difference to CloneParameter2List() is
that it does not actually copy the whole param object
This function will not add a parameter to the list, if one
with the same name already exists

Parameters

param - pointer to an existing PARAM object

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	PARAM	PARAM

Function

PARAM::PARAM(char *name, char *section, char *type, char *value, char *my_defaultvalue, char *my_lowrange, char *my_highrange, char *my_comment)

Purpose

Constructs and initializes a parameter object

Parameters

self-explanatory

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	PARAM()	PARAM

Function

PARAM::PARAM()

Purpose

The constructor for the PARAM object

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	PARAM(char *paramstring)	PARAM

Function

PARAM::PARAM(char *paramstring)

Purpose

Constructs and initializes a parameter object, based upon
a parameter string, as defined in the project outline

Parameters

char *paramstring

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	PARAMLIST()	PARAMLIST

Function

PARAMLIST::PARAMLIST()

Purpose

the constructor for the PARAMLIST object

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	ParseParameter(char *line, int lengt	PARAM

Function

int PARAM::ParseParameter(char *line, int length)

Purpose

This routine is called by coremessage->ParseMessage()
it parses the provided ASCII parameter line and initializes
the PARAM object accordingly, i.e., it fills in values, name, section name, type, etc.

Parameters

line - pointer to the ASCII parameter line
length - length of this parameter line

Returns

ERRPARAM_INVALIDPARAM if the parameter line is invalid, or
ERRPARAM_NOERR

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	SetListPtr(TList *temp_list)	void PARAM

Function

void PARAM::SetListPtr(TList *temp_list)

Purpose

Sets this parameter's list of values

Parameters

pointer to the list

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	SetName(char *src)	PARAM

Function

void PARAM::SetName(char *src)

Purpose

sets the name of the parameter
Note: parameter names are case insensitive

Parameters

char pointer to the name

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	SetSection(char *src)	PARAM

Function

void PARAM::SetSection(char *src)

Purpose

sets the section name of the parameter

Parameters

char pointer to the section name

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	SetType(char *src)	PARAM

Function

void PARAM::SetType(char *src)

Purpose

sets the type of the parameter

Parameters

char pointer to the type name

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UParameter.cpp	SetValue(char *src)	PARAM

Function

void PARAM::SetValue(char *src)

Purpose

sets the (first) value of the parameter

Parameters

char pointer to the value

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	~STATE()	STATE

Function

STATE::~STATE()

Purpose

the destructor for the STATE object

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	~STATELIST()	STATELIST

Function

STATELIST::~STATELIST()

Purpose

The destructor for the STATELIST object. It deletes both all STATE objects in the list and also the list object itself

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	AddState2List(char *statestring)	STATELIST

Function

void STATELIST::AddState2List(char *statestring)

Purpose

adds a new state to the list of states
if a state with the same name already exists,
it updates the currently stored state with the provided values

Parameters

paramstring - ASCII string, as defined in project description,
defining this new state

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	AddState2List(STATE *state)	STATELIST

Function

void STATELIST::AddState2List(STATE *state)

Purpose

Adds a new state to the list of states
It physically copies the STATE object (the specified STATE
object can then be freed elsewhere)
if a state with the same name already exists,
it updates the currently stored state with the provided values

Parameters

state - pointer to an existing STATE object

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	ClearStateList()	STATELIST

Function

void STATELIST::ClearStateList()

Purpose

clears and frees all entries in the state list

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	ConstructStateLine()	STATE

Function

int STATE::ConstructStateLine()

Purpose

Construct a state line, based upon the current values in the STATE object

Parameters

N/A

Returns

always ERRPARAM_NOERR

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	DeleteState(char *name)	STATELIST

Function

void STATELIST::DeleteState(char *name)

Purpose

deletes a state of a given name in the list of states
it also frees the memory for this particular state
it does not do anything, if the state does not exist

Parameters

name - name of the state

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	get_argument(int ptr, char *buf, cha	STATE

Function

int STATE::get_argument(int ptr, char *buf, char *line, int maxlen)

Purpose

parses the state line that is being sent in the core
communication
it returns the next token that is being delimited by either a ' ' or '='

Parameters

ptr - index into the line of where to start
buf - destination buffer for the token
line - the whole line maxlen - maximum length of the line

Returns

the index into the line where the returned token ends

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetBitLocation()	STATE

Function

int STATE::GetBitLocation()

Purpose

Returns this state's bit location in the state vector

Parameters

N/A

Returns

this state's bit location

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetByteLocation()	STATE

Function

int STATE::GetByteLocation()

Purpose

Returns this state's byte location in the state vector

Parameters

N/A

Returns

this state's byte location

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetLength()	STATE

Function

int STATE::GetLength()

Purpose

Returns this state's length

Parameters

N/A

Returns

this state's length

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetListPtr()	TList *STATELIST

Function

TList *STATELIST::GetListPtr()

Purpose

Returns a pointer to this state's list of values

Parameters

N/A

Returns

pointer to the list

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetName()	char *STATE

Function

char *STATE::GetName()

Purpose

Returns the name of this state

Parameters

N/A

Returns

a pointer to the state's name

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetNumStates()	STATELIST

Function

int STATELIST::GetNumStates()

Purpose

Returns the number of states in the list

Parameters

N/A

Returns

the number of states

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetStateLine()	char *STATE

Function

char *STATE::GetStateLine()

Purpose

Returns a state line in ASCII format
This state line is constructed, based upon the current values in the STATE object

Parameters

N/A

Returns

a pointer to the state line

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetStateListPtr()	STATELIST *STATEVECTOR

Function

STATELIST *STATEVECTOR::GetStateListPtr()

Purpose

returns a ptr to the list of states describing the state vector

Parameters

N/A

Returns

pointer to the list of states

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetStatePtr(char *name)	STATE *STATELIST

Function

STATE *STATELIST::GetStatePtr(char *name)

Purpose

given a state's name, returns the pointer to a STATE object

Parameters

name - name of the state

Returns

pointer to a STATE object or NULL, if no state with this name exists in the list

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetStatePtr(int idx)	STATE *STATELIST

Function

STATE *STATELIST::GetStatePtr(int idx)

Purpose

given an index (0..GetListCount()-1), returns the pointer to a STATE object

Parameters

idx - index of the state

Returns

pointer to a STATE object or NULL, if the specified index is out of range

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetStateValue(char *statename)	unsigned short STATEVECTOR

Function

unsigned short STATEVECTOR::GetStateValue(char *statename)

Purpose

returns a state's value from the state vector

Parameters

statename - the name of a state

Returns

the value of the state
0 on error (e.g., state not found)

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetStateValue(int byteloc, int bitloc,	unsigned short STATEVECTOR

Function

unsigned short STATEVECTOR::GetStateValue(int byteloc, int bitloc, int length)

Purpose

returns a state's value, based upon the state's location and size

Parameters

byteloc ... location of the byte of the state
bitloc ... location of the bit of the state
length ... length of the state

Returns

ERRSTATEVEC_NOSTATE on error (e.g., location outside range) the value of the state otherwise

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetStateVectorLength()	STATEVECTOR

Function

int STATEVECTOR::GetStateVectorLength()

Purpose

returns the length of the state vector in bytes

Parameters

N/A

Returns

the length of the state vector in bytes

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetStateVectorPtr()	BYTE *STATEVECTOR

Function

BYTE *STATEVECTOR::GetStateVectorPtr()

Purpose

returns a BYTE ptr to the state vector data

Parameters

N/A

Returns

a pointer to the state vector data

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	GetValue()	unsigned short STATE

Function

unsigned short STATE::GetValue()

Purpose

Returns this state's value

Parameters

N/A

Returns

this state's value

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	Initialize_StateVector()	STATEVECTOR

Function

void STATEVECTOR::Initialize_StateVector()

Purpose

assignes locations in the state vector for each state; in addition, it initializes the state vector with the values of each state in the list

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	ParseState(char *line, int statelength	STATE

Function

int STATE::ParseState(char *line, int statelength)

Purpose

This routine is called by coremessage->ParseMessage()
it parses the provided ASCII state line and initializes
the STATE object accordingly, i.e., it fills in value, name, byte location, bit location, etc.

Parameters

line - pointer to the ASCII state line length - length of this state line

Returns

ERRSTATE_INVALIDSTATE if the state line is invalid, or ERRSTATE_NOERR

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	SetBitLocation(int loc)	STATE

Function

void STATE::SetBitLocation(int loc)

Purpose

Sets this state's bit location in the state vector

Parameters

this state's bit location

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	SetByteLocation(int loc)	STATE

Function

void STATE::SetByteLocation(int loc)

Purpose

Sets this state's byte location in the state vector

Parameters

this state's byte location

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	SetStateValue(char *statename, unsi	STATEVECTOR

Function

int STATEVECTOR::SetStateValue(char *statename, unsigned short stateval)

Purpose

sets a state's value in the state vector

Parameters

byteloc ... location of the byte of the state
bitloc ... location of the bit of the state
length ... length of the state
value ... value of the state

Returns

ERRSTATEVEC_NOSTATE on error (e.g., location out of range)
ERRSTATEVEC_NOERR otherwise

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	SetStateVector(BYTE *src, int lengt	STATEVECTOR

Function

void STATEVECTOR::SetStateVector(BYTE *src, int length, STATELIST *src_list)

Purpose

sets the content of a whole state vector

Parameters

src - pointer to the source state vector content
length - length of the state vector in bytes
src_list - pointer to the list of states describing this state vector

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	SetValue	STATE

Function

void STATE::SetValue(unsigned short new_value)

Purpose

Sets this state's value

Parameters

the new value for this state

Returns

N/A

Group	Responsible	EMailAddress

ModuleName	FunctionName	ClassName
UState.cpp	STATE()	STATE

Function

STATE::STATE()

Purpose

the constructor for the STATE object

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UState.cpp	STATELIST	STATELIST

Function

STATELIST::STATELIST()

Purpose

the constructor for the STATELIST object

Parameters

N/A

Returns

N/A

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UStatus.cpp	GetCode()	STATUS

Function

int STATUS::GetCode()

Purpose

Parameters

Returns

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UStatus.cpp	GetStatus()	char *STATUS

Function

char *STATUS::GetStatus()

Purpose

Parameters

Returns

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org

ModuleName	FunctionName	ClassName
UStatus.cpp	ParseStatus(char *line, int length)	STATUS

Function

int STATUS::ParseStatus(char *line, int length)

Purpose

Parameters

Returns

Group	Responsible	EMailAddress
Albany	Gerv	schalk@wadsworth.org