$this

The WellCAD's build-in data source containing information about objects of the borehole document. Currently there are four tables accessible: *Logs*, *Layers*, *Header* and *Borehole*

Remarks:

The ODBC engine must be in "saving mode", i.e. the script should be opened via the "Save Database" menu.

The *Logs* table contains the fields:

Name (text) the log title*;*
Type (text) the log type (WellLog, MudLog, ...);
TopDepth (number) top depth of the log in [m];
BottomDepth (number) bottom depth of the log in [m];
Dictionary (text) dictionary name used by Lithology Log, Analysis Log, Percentage Log, ...;
Dimension (number) the number of data points in a data trace (Image Log, FWS Log, ...)
Increment (number) the time sampling rate used in FWS logs.

The *Layers* table contains the fields:

Name (text) the annotation layer title;
Type (text) is always set to "AnnotationLog".

The *Header* table contains one record for each dynamic header text item:

Name (text) the dynamic header text item title;
Value (text) the dynamic header item text.

The *HeaderTables* table contains the fields:

Name (text) the identifying name of a static header table object;

The *Borehole* table contains the following fields:

TopDepth (number) the borehole document top depth in current depth units;
BottomDepth (number) the borehole document bottom depth in current depth units;
DepthUnit (text) the current borehole document depth unit ("m" or "ft").

Example:

```
# select all fields from the build-in Logs table
$logs = SELECT * FROM Logs using $this

# select all fields from the build-in Layers table
$layers = SELECT * FROM Layers using $this

# get the text displayed next to the “Well:” in the WellCAD header
$uwirset = SELECT Value FROM Header WHERE Name = 'WELL:' using $this

# get all the fields from a static header table object named “Tops”
$Tops = SELECT * FROM HeaderTables WHERE Name = 'Tops' using $this

# get the borehole document top, bottom and depth unit string
# update a row in the RunVersion table with this information
$DocInfo = SELECT * FROM Borehole using $this
UPDATE RunVersion SET versTop = $DocInfo.TopDepth,
+ versBase = $DocInfo.BottomDepth, versDepthUnit = '$DocInfo.DepthUnit'
+ WHERE versID = $VERSID
```

$DepthUnit

A global variable containing the depth unit used when exchanging depth related data with the database.

Remarks:

The value can be “M” or “FT”.

Example:

```
$DepthUnit = m
```

$MaxStringSize

A global variable containing the buffer size for each string field in the record set.

Remarks:

The value can be any interger value between 16 and 65535.

Example:

```
$MaxStringSize = 65535
```

$DocumentTitle

A global string variable containing the title of the imported document in WellCAD.

Example:

```
$Title = SELECT wellHeaderID+'_'+runHeaderID
+ AS Title
+ FROM Wells, Runs
+ WHERE Wells.wellID = $WellID
+ AND Runs.runID = $RunID

$DocumentTitle = $Title.Title
CLOSE($Title)
```

$CreateEmptyLogs

A global string variable containing allowing the import of empty logs when set to true (or yes).

Example:

```
$CreateEmptyLogs = true
```

OpenDatabase (*DSNName (text)* , *UserName (text)* , *bExclusive (bool)* , *bReadOnly (bool)* , *Password (text)*)

Opens a ODBC data source. The data source must have been created in the ODBC applet of the operation system's control panel before.

Parameter:

DSNName	The ODBC data source name as labeled in the ODBC data source configuration in the control panel.
UserName	The user name used to login to the DSN.
bExclusive	If set to TRUE the database opens in exclusive mode.
bReadOnly	If set to TRUE the database in read-only mode.
Password	The user's password to gain access to the DSN.

Remarks:

The functions returns a variable of type DSN used afterwards for all ODBC operations.

The bReadOnly flag should be set to TRUE for scripts performing a data load operation (database to WellCAD). When storing data from WellCAD into the database the *bReadOnly* parameter must be set to FALSE.

Example:

```
# open data source for storing
$Server = Test
$User =
$Password =
$dsn = OpenDatabase($Server, $User, false, false, $Password)
. . .
CLOSE($dsn)

# open data source for loading
$dsn = OpenDatabase($Server, $User, false, true, $Password)
. . .
CLOSE($dsn)
```

OpenDatabaseDyn (*DriverName (text)* , *DatabaseName (text)* , ...)

Registers an ODBC source dynamically using the information from the parameter list. After registration of the function it opens the ODBC data source.

Parameter:

DriverName	A string specifying the driver name as presented to the user in the ODBC applet of the operation system's control panel. Eg. "Microsoft Access Driver (*.mdb)" -or- "Excel Files (*.xls)" -or- "SQL Server" -or- any other string depending on which database driver you are using.
DatabaseName	An ODBC connection string will be build from the items you pass in the parameter list. Special key words depending on the database driver you are using have to be set. The syntax for each string passed in the parameter list is *KEYWORD=description.*
...	

Remarks:

An ODBC connection string will be build from the items you pass in the parameter list. Special key words depending on the database driver you are using have to be set. For Microsoft Access you have to pass at least the DSN and DBQ keywords as shown in the example below. If you call the OpenDatabaseDyn() function without parameters a dialog box pops up which prompts the user for the necessary information. After the database is closed the ODBC source is unregistered.

Example:

```
# register and open MS Access data source

$driver = Microsoft Access Driver (*.mdb)
$server = DSN=Demo
$dbq = DBQ=C:\\Temp\\Database\\Demo.mdb
$dsn = OPENDATABASEDYN($driver, $server, $dbq)
. . .
CLOSE($dsn)
```

RegisterDatabaseDyn (*DriverName (text)* , *DatabaseName (text)* , ...)

Registers an ODBC source dynamically using the information from the parameter list.

Parameter:

DriverName	A string specifying the driver name as presented to the user in the ODBC applet of the operation system's control panel. Eg. "Microsoft Access Driver (*.mdb)" -or- "Excel Files (*.xls)" -or- "SQL Server" -or- any other string depending on which database driver you are using.
DatabaseName	An ODBC connection string will be build from the items you pass in the parameter list. Special key words depending on the database driver you are using have to be set. The syntax for each string passed in the parameter list is *KEYWORD=description.*
...	

Remarks:

This funtion registers an ODBC source using the information from a ODBC string provided. It does not open the database. An additional call of the OpenDatabase function is still necessary.
The ODBC connection string will be build from the items you pass in the parameter list. Special key words depending on the database driver you are using have to be set. For Microsoft Access you have to pass at least the DSN and DBQ keywords as shown in the example below. If you call the RegisterDatabaseDyn() function without parameters a dialog box pops up which prompts the user for the necessary information.

Example:

```
# register MS Access data source

$driver = Microsoft Access Driver (*.mdb)
$server = DSN=Demo
$dbq = DBQ=C:\\Temp\\Database\\Demo.mdb
REGISTERDATABASEDYN($driver, $server, $dbq)
$dbname = Demo
$user =
$dsn = OpenDatabase($dbname, $user, false, true)
. . .
CLOSE($dsn)
```

UnRegisterDatabaseDyn (*DriverName (text)* , *DatabaseName (text)* , ...)

Unregisters an ODBC source dynamically using the information from the parameter list.

Parameter:

DriverName	A string specifying the driver name as presented to the user in the ODBC applet of the operation system's control panel. Eg. "Microsoft Access Driver (*.mdb)" -or- "Excel Files (*.xls)" -or- "SQL Server" -or- any other string depending on which database driver you are using.
DatabaseName	An ODBC connection string will be build from the items you pass in the parameter list. Special key words depending on the database driver you are using have to be set. The syntax for each string passed in the parameter list is *KEYWORD=description.*
...	

Remarks:

This funtion unregisters an ODBC source using the information from a ODBC string provided. The ODBC connection string will be build from the items you pass in the parameter list. Special key words depending on the database driver you are using have to be set. For Microsoft Access you have to pass at least the DSN.

Example:

```
. . .
# unregister MS Access data source
$driver = Microsoft Access Driver (*.mdb)
$server = DSN=Demo
UNREGISTERDATABASEDYN($driver, $server)
. . .
CLOSE($dsn)
```

Close (*$DSN (DSN)*)

Closes an opened ODBC data source.

Parameter:

$DSN Contains the ODBC data source to close.

Remarks:

Example:

```
# open data source for storing
$Server = Test
$User =
$Password =
$dsn = OpenDatabase($Server, $User, false, false, $Password)
. . .
CLOSE($dsn)

# open data source for loading
$dsn = OpenDatabase($Server, $User, false, true, $Password)
. . .
CLOSE($dsn)
```

CreateLogTable (*Log Title (text)* , *Log Table Name (text)*)

Creates a new table and inserts the log type specific fields for the log identified by its title.

Parameter:

Log Title	The log title as it has been set in the borehole document.
Log Table Name	The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The fields inserted depend on the WellCAD log that is identified by the first parameter.

Example:

```
$Logs = SELECT * FROM Logs using $this

# insert a new row into Log table
INSERT INTO Log (VersionID, Title, LogType, TopDepth, BottomDepth)
+ VALUES ($VERSID,'$Logs.Name','$Logs.Type',$Logs.TopDepth,$Logs.BottomDepth)

# get a unique LogTableName based on our unique LogID
$log = SELECT LogID FROM Log WHERE VersionID = $VERSID AND Title = '$Logs.Name'
$LogTableName = L$log.LogID

# update the Logs row
UPDATE Log SET LogTableName = '$LogTableName' WHERE LogID = $log.LogID
close($log)

# create the LogTable
CREATELOGTABLE($Logs.Name,$LogTableName)
```

CreateLayerTable (*Layer Title (text)* , *Layer Table Name (text)*)

Creates a new table and inserts the fields for the annotation layer identified by its title.

Parameter:

Layer Title	The layer title as it has been set in the borehole document.
Layer Table Name	The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. Default fields are inserted for the layer identified by the first parameter.

Example:

```
$Layers = SELECT * FROM Layers using $this

# insert a new row into Layer table
INSERT INTO Layer (VersionID, Title)
+ VALUES ($VERSID,'$Layers.Name')

# get a unique LayerTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$LayerTableName = L$layer.LayerID

# update the Layer row
UPDATE Layer
+ SET LayerTableName = '$LayerTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the LayerTable
CREATELAYERTABLE($Layers.Name,$LayerTableName)
```

CreateHeaderTableTable (*Table Title (text)* , *Database Table Name (text)*)

Creates a new database table and inserts the fields corresponding to the columns in the static table object in the WellCAD header.

Parameter:

Table Title	The table title as it identifies the table in the borehole document header.
Database Table Name	The name of the new database table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The number and names of fields inserted depend on the number and names of columns in the WellCAD header table object.

Example:

```
# get all the fields from a static header table object named “Tops”
$Tops = SELECT * FROM HeaderTables WHERE Name = 'Tops' using $this

# the new table name used in the database
$DBTableName = FormationTops

# create the Header Table Table
CREATEHEADERTABLETABLE($Tops.Name, $DBTableName)
```

LoadHeaderForm (*Header File Path (text)*)

LoadHeaderForm (*Header Data Field (record set), bBLOB (bool), bReplace (bool), bTrailer (bool)*)

Loads a WellCAD header form and attaches it to the new document.

Parameter:

Header File Path	The complete path to a *.wch or *.wah file.
Header Data Field	A record set containing the header in binary or ascii format stored in a single field.
bBLOB	A flag (“TRUE” or “FALSE”) indicating whether the stored data must be read as long binary object (BLOB) or as text (ascii).
bReplace	Specifies whether the current header / trailer attached to the borehole document is to be replaced (flag set to TRUE) or not. As there is always a default header attached to a new document, the first call of the LoadHeaderForm function should have the *bReplace* flag set to TRUE.
bTrailer	Specifies whether the form loaded is a header (placed at the top of the document) or a trailer (displayed below the borehole document).

Remarks:

The second version of the function looks for the fieldname *HeaderForm (longbinary* or *text)* and loads the stored data according to the second parameter.

Example:

```
# Load header form file
$Form = SELECT HeaderForm, descID
+ FROM CoreDescription
+ WHERE ID = $DESCID
LOADHEADERFORM($Form, TRUE)
CLOSE($Form)

# Load header form binary
$Form = SELECT HeaderForm, descID
+ FROM CoreDescription
+ WHERE ID = $DESCID
LOADHEADERFORM('C:\TEMP\HEADER\Coredesc.wch')
CLOSE($Form)
```

StoreHeaderForm (*Header Data Field (record set)*)

Stores the current WellCAD header form to the database.

Parameter:

Header Data Field A record set identifying the field to store the header form in.

Remarks:

The function looks for the fieldname *HeaderForm (longbinary)* and stores the current WellCAD header as a long binary object (BLOB).

Example:

```
$Store = SELECT HeaderForm, descID
+ FROM CoreDescription
+ WHERE descID = $DESCID
STOREHEADERFORM($Store)
CLOSE($Store)
```

LoadHeader (*Header Data (record set)*)

Fills a WellCAD header form by matching field names and dynamic text field titles of the header.

Parameter:

Header Data A record set containing all the

Remarks:

Each field name or alias of the record set is matched against a dynamic text field title of the current WellCAD header form. If a correspondence has been found the information is loaded to the dynamic text field. Not existing header words will be skipped.

Example:

```
# Load header information from Project and Well table
$Header = SELECT wellUWI as 'WELL:',
+ projectTitle AS 'PROJECT:'
+ projectCreateDate AS 'PROJECT CREATE DATE:',
+ wellTitle AS 'Well Name',
+ wellCreateDate AS 'WELL CREATE DATE:',
+ wellLocation AS 'LOCATION:'
+ FROM Project, Well
+ WHERE Project.projectID = $ProjectID
+ AND Well.wellID = $WellID
LOADHEADER($Header)
CLOSE($Header)
```

StoreHeader (*Header Data (record set)* , *Add Fields...*)

StoreHeader (*Header Data (record set)* , *Existing Data (record set)* , *Add Fields...*)

Creates a new database table and inserts the fields to hold the data of the plug symbol.

Parameter:

Header Data	A record set identifying the fields to store the WellCAD header information to.
Add Fields	Additional data, not contained in the current header form, can be assigned to fields of the *Header Data* record set.
Existing Data	A record set used to apply a conditional testing if a new row must be added to the database table or if an existing row can be updated.

Remarks:

Each dynamic text field title of the current WellCAD header form is matched against the field names of the record set. If a match has been found the header information is stored to the assigned field. If a row in the database table can be updated or if a new one needs to be inserted can be tested with the *Existing Data* parameter. Data which is not contained in the header form can be passed using the *Add Fields* .

Example:

```
# store header information into the well table
# check if same well (wellUWI) already exists in the project
# update additional field with project ID
$Store = SELECT projectID,
+ wellUWI AS 'WELL:',
+ wellTitle AS 'Well Name',
+ wellCreateDate AS 'WELL CREATE DATE:',
+ wellLocation AS 'LOCATION:'
+ FROM Well
$Exist = SELECT projectID, wellUWI AS 'WELL:' FROM Well
+ WHERE projectID = $ProjectID
# and store
STOREHEADER($Store, $Exist, projectID = $ProjectID)
CLOSE($Store)
CLOSE($Exist)
```

LoadHeaderTable (*TableName (text), ColumnData (record set)*)

Fills a static header table object in the WellCAD header form by matching the TableName parameter with the name identifying the object in the header.

Parameter:

TableName	The identifying name of the table object.
ColumnData	A record set containing the data that should be displayed in the table cells

Remarks:

If the *TableName* parameter matches the name of a static header table in the WellCAD header form the data contained in the *ColumnData* record set is copied to that table. Field names or their alias will be used as column headers.

Example:

```
# Load information from Tops table to populate the
# Formation table object in WellCAD
$Tops = SELECT * FROM Tops
LOADHEADERTABLE(Formation, $Tops)
CLOSE($Tops)
```

StoreHeaderTable (*TableName (text), ColumnData (record set)*)

Stores the information of a static header object contained in the WellCAD header form

Parameter:

TableName	The identifier of the static header object within the WellCAD header form.
ColumnData	The destination record set to hold the data from the static table object.

Remarks:

The database field names or alias must match the column names of the header table object to transfer the data successfully.

Example:

```
# store header information from Formations object into the Tops table
$Store = SELECT * FROM Tops
STOREHEADERTABLE(Formations,$Store)
CLOSE($Store)
```

LoadLog (*Log Info (record set)* , *Log Data (record set)*)

Identifies the log type, creates a new log in WellCAD and downloads the log data from the database.

Parameter:

Log Info	A record set containing the information about the WellCAD log type, the log title and trace increment (used for FWS log only).
Log Data	The record set containing the data

Remarks:

Log Info must contain the field names *LogType (text)* and *Title (text)*. The fieldname *TraceIncrement (float)* is optional, but should be set when FWS logs are loaded.
Log Data must contain the depth and data fields according to each log type (see the specific Load... functions for each log type.).
Valid *LogType*s are: WellLog, MudLog, DepthLog, IntervalLog, CommentLog, StrataLog, StackingPatternLog, ImageLog, ImageFloatLog, RgbLog, VspLog, FwsLog, OleLog, BioLog, EngineeringLog, StructureLog, Polarlog, CrossSectionLog, LithologicalLog, AnalysisLog, Percentagelog, CoreDescLog

Example:

```
# load info from Log table
$LOGINFO = SELECT * FROM Log WHERE descID = $DESCID
# load all logs with the same descID
LOOPON($LOGINFO)
     $LOGDATA = SELECT * FROM $LOGINFO.LogTableName
     LOADLOG($LOGINFO,$LOGDATA)
     CLOSE($LOGDATA)
ENDLOOP
CLOSE($LOGINFO)
```

StoreLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the WellCAD log data. The fields (and required field names) are generated using the *CreateLogTable* function.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the log table
CREATELOGTABLE($Logs.Name,$LogTableName)

# add additional columns
ALTER TABLE $LogTableName
+ ADD RunNumber varchar(255), LogDate varchar(255)

# store data to the LogTableName table,
# add additional information to the RunNumber and LogDate fields
$log = SELECT * FROM $LogTableName
STORELOG($Logs.Name,$log, RunNumber = $Run, LogDate = $Date)
CLOSE($log)
. . .
```

LoadWellLog (*Log Title (text)* , *Log Data (record set)*)

LoadWellLog (*Log Data (record set)*)

Creates one or more new Well Logs in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Well Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain at least a *Depth (float)* and a data column. *Depth* must be the first field and should contain sorted constant sampling rate depth values. The second field is taken as data column. Each additional field is interpreted as data column as well.
If no log title is passed as first parameter or if more than one data column exists the field names are taken as log titles.

Example:

```
. . .
# load a single WellLog
$LOGDATA = SELECT * FROM $VERS.datatable
LOADWELLLOG('GR',$LOGDATA)
. . .

-OR-

. . .
# load all WellLogs
LOOPON($LOGINFO)
     $LOGDATA = SELECT Depth, Data AS '$LOGINFO.Title'
                FROM $LOGINFO.LogTableName
     LOADWELLLOG($LOGDATA)
     CLOSE($LOGDATA)
ENDLOOP
CLOSE($LOGINFO)
. . .
```

StoreWellLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Well Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth (float)* and *Data (float)* (or *Value* or the exact log title).
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $LogTableName (Depth float, Data float null)

# add additional columns for additional fields
ALTER TABLE $LogTableName
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $LogTableName
STOREWELLLOG($Logs.Name, $log, RunNumber = $Run, HoleID = &UWI)

# done
CLOSE($log)
. . .
```

LoadMudLog (*Log Title (text)* , *Log Data (record set)*)

LoadMudLog (*Log Data (record set)*)

Creates one or more new Mud Logs in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Mud Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain at least a *Depth (float)* and a data column. *Depth* must be the first field and should contain sorted depth values. The second field is taken as data column. Each additional field is interpreted as data column as well.
If no log title is passed as first parameter or if more than one data column exists the field names are taken as log titles.

Example:

```
. . .
# load a single MudLog
$logdata = SELECT * FROM $vers.datatable
LOADMUDLOG('Progress',$logdata)
. . .

-OR-
. . .
# load all MudLogs
LOOPON($loginfo)
     $logdata = SELECT Depth, Data AS '$loginfo.Title'
                FROM $loginfo.LogTableName
     LOADMUDLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreMudLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Mud Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth (float)* and *Data (float)* (or *Value* or the exact log title).
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $LogTableName (Depth float, Data float null)

# add additional columns for additional fields
ALTER TABLE $LogTableName
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * From $LogTableName
STOREMUDLOG($Logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadRgbLog (*Log Title (text)* , *Log Data (record set)*)

LoadRgbLog (*Log Data (record set)*)

Creates a new RGB Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the RGB Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain a *Depth (float)* and a *Data (longbinary)* column. *Depth* must be the first field and should contain sorted depth values of constant sampling rate. The second column must contain a trace of RGB values stored as binary object (BLOB).

Example:

```
. . .
# load a single RGBLog
$logdata = SELECT * FROM $vers.datatable
LOADRGBLOG('CorePhoto',$logdata)
. . .

-OR-
. . .
# load all RGBLogs
LOOPON($loginfo)
     $logdata = SELECT Depth, Data AS '$loginfo.Title'
                FROM $loginfo.LogTableName
     LOADRGBLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreRgbLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified RGB Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth (float)* and *Data (longbinary)*. A complete trace of RGB values will be store as binary object (BLOB) for each depth.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $LogTableName (Depth float, Data longbinary)

# add additional columns for additional fields
ALTER TABLE $LogTableName
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $LogTableName
STORERGBLOG($Logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadImageLog (*Log Title (text)* , *Log Data (record set)*)

LoadImageLog (*Log Data (record set)*)

Creates a new Image Logs in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Image Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must at least contain a *Depth (float)* and a *Data (longbinary)* column. *Depth* must be the first field and should contain sorted depth values at a constant sampling rate. If only a *Depth* and a *Data* column can be found the data is read as a trace of unsigned integer values stored as binary object (BLOB) for each depth. If a series of fields is passed they will be read as an array of unsigned integer data (one value from each field).

Example:

```
. . .
# load a single ImageLog
$logdata = SELECT * FROM $vers.datatable
LOADIMAGELOG('FMI',$logdata)
. . .

-OR-
. . .
# load all ImageLogs
LOOPON($loginfo)
     $logdata = SELECT Depth, Data
                FROM $loginfo.LogTableName
     LOADIMAGELOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreImageLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Image Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth (float)* and *Data (longbinary)*. A complete trace of Integer values will be stored as binary object (BLOB) for each depth.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $LogTableName (Depth float, Data longbinary)

# add additional columns for additional fields
ALTER TABLE $LogTableName
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $LogTableName
STOREIMAGELOG($Logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadImageFloatLog (*Log Title (text)* , *Log Data (record set)*)

LoadImageFloatLog (*Log Data (record set)*)

Creates a new Image Log (float) in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Image Log (float)
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must at least contain a *Depth (float)* and a *Data (float)* column. *Depth* must be the first field and should contain sorted depth values at a constant sampling rate. If only a *Depth* and a *Data* column can be found the data is read as a trace of floating point values stored as binary object (BLOB) for each depth. If a series of fields is passed they will be read as an array of floating point data (one value from each field).

Example:

```
. . .
# load a single ImageFloatLog
$LOGDATA = SELECT * FROM $vers.datatable
LOADIMAGEFLOATLOG('FMI',$LOGDATA)
. . .

-OR-
. . .
# load all ImageLogs
LOOPON($loginfo)
     $logdata = SELECT Depth, Data
                FROM $loginfo.LogTableName
     LOADIMAGEFLOATLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreImageFloatLog (*Log Name (text)* , *Data Table (record set), Add Fields*)

Stores the data of the specified Image Log (float) to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth* and *Data*. A complete trace of floating point values will be stored as binary object (BLOB) for each depth.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (Depth float, Data longbinary)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STOREIMAGEFLOATLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadCommentLog (*Log Title (text)* , *Log Data (record set)*)

LoadCommentLog (*Log Data (record set)*)

Creates one or more new Comment Logs in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Comment Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain at least a *TopDepth (float)*, *BottomDepth (float)* and a data column. All fields other than *TopDepth* and *BottomDepth* contained in the record set are read as text information. One comment log will be created for each column.

Example:

```
. . .
# load a single CommentLog
$logdata = SELECT * FROM $vers.datatable
LOADCOMMENTLOG('Description',$logdata)
. . .

-OR-
. . .
# load all CommentLogs
LOOPON($loginfo)
     $logdata = SELECT TopDepth, BottomDepth, Comment
                FROM $loginfo.logtablename
     LOADCOMMENTLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreCommentLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Comment Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *TopDepth (float), BottomDepth (float)* and *Comment (text)*.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (TopDepth float, BottomDepth float,
+ Comment longtext)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STORECOMMENTLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadStructureLog (*Log Title (text)* , *Log Data (record set),* *Dictionary (record set)*)
LoadStructureLog (*Log Title (text)* , *Log Data (record set),* *Dictionary (text)*)
LoadStructureLog (*Log Title (text)* , *Log Data (record set)*)
LoadStructureLog (*Log Data (record set)*)

Creates a new Structure Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Structure Log
Log Data	The record set containing the data
Dictionary	The path to a dictionary file or a record set containing the tad poles

Remarks:

The *Log Data* record set must contain a *Depth (float)*, *Azimuth (float)* and *Tilt(float)* column. *Code (int)* and *Description (text)* are optional fields. The dictionary parameter specifies the file or record set that contains the tadpole symbols the *Code* refers to.

Example:

```
. . .
# load a single StructureLog
$logdata = SELECT * FROM $vers.datatable
LOADSTRUCTURELOG('Structure',$logdata, 'C:\TEMP\TadPoles.tad)
. . .

-OR-
. . .
# load all Structure Logs
LOOPON($loginfo)
     $logdata = SELECT Depth, Azimuth, Tilt, Code
                FROM $loginfo.logtablename
     LOADSTRUCTURELOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreStructureLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Structure Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth*, *Azimuth, Title*. *Code* and *Description* are optional fields.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (Depth float, Azimuth float, Tilt float,
+ Code int, Description longtext)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STORESTRUCTURELOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

StoreFractureDictionary (*Data Table (record set), Dictionary Path (text)*)

Stores the contents of the specified Tadpole Dictionary file (*.tad) to the database.

Parameter:

Data Table	The record set that will store the dictionary data.
Dictionary Path	The path and file name of the *.tad file to be stored.

Remarks:

The *Data Table* record set must contain all fields necessary to store the lithology pattern data contained in the *.lth file. The fields required are: *Code (int), Shape (int), TailSize (lonbinary), Color (int), Width (int), Height (int)*, *Description (text)*, *DictionaryName (text)*.

Example:

```
. . .
# store dictionary
$patterns = SELECT * FROM $dictionaries
STOREFRACTUREDICTIONARY($log, 'C:/Temp/Default.tad')
# done
CLOSE($patterns)
. . .
```

LoadLithoLog (*Log Title (text)* , *Log Data (record set)* , *Dictionary (record set)*)
LoadLithoLog (*Log Title (text)* , *Log Data (record set)* , *Dictionary (Text)*)
LoadLithoLog (*Log Title (text)* , *Log Data (record set)*)
LoadLithoLog (*Log Data (record set)*)

Creates a new Lithology Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Mud Log
Log Data	The record set containing the data
Dictionary	A record set or the path to a file containing the lithology patterns.

Remarks:

The *Log Data* record set must contain at least a *TopDepth (float)* and *BottomDepth (float)* column. The fields *Code (varchar(255))*, *Hardness (float)*, *Position (float)*, *TopContact (longtext)* and *BottomContact (longtext)* are optional. The dictionary parameter specifies the file or record set that contains the lithology patterns the *Code* refers to.

Example:

```
. . .
# load a single Lithology Log
$logdata = SELECT * FROM $vers.datatable
LOADLITHOLOG('Lithology',$logdata,'C:\TEMP\Structure.lth)
. . .

-OR-
. . .
# load all Lithology Logs
LOOPON($loginfo)
     $logdata = SELECT TopDepth, BottomDepth, Code, Hardness, TopContact
                FROM $loginfo.logtablename
     LOADLITHOLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreLithoLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Lithology Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *TopDepth , BottomDepth, Code*. Optional fields are *Hardness, Position, TopContact, BottomContact.*
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (TopDepth float, BottomDepth float,
+ Code varchar(255), TopContact longtext

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STORELITHOLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

StoreLithoDictionary (*Data Table (record set), Dictionary Path (text)*)

Stores the contents of the specified Litho Dictionary file (*.lth) to the database.

Parameter:

Data Table	The record set that will store the dictionary data.
Dictionary Path	The path and file name of the *.lth file to be stored.

Remarks:

The *Data Table* record set must contain all fields necessary to store the lithology pattern data contained in the *.lth file. The fields required are: *Code (text), Name (text), Symbol (lonbinary), Width (int), Height (int),* Optional fields are *Color (int), DictionaryName (text), Repeatable (int), SortCode (int).*

Example:

```
. . .
# store dictionary
$patterns = SELECT * FROM $dictionaries
STORELITHODICTIONARY($log, 'C:/Temp/Default.lth')
# done
CLOSE($patterns)
. . .
```

LoadFullwaveLog (*Log Title (text)* , *Log Data (record set)* , *TraceIncrement (text)*)
LoadFullwaveLog (*Log Title (text)* , *Log Data (record set)*)
LoadFullwaveLog (*Log Data (record set)*)

Createsa new FWS Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Mud Log
Log Data	The record set containing the data
TraceIncrement	The time sampling rate of the full wave form trace in µs.

Remarks:

The *Log Data* record set must contain a *Depth (float)* and a *Data (longbinary)* column with the sonic trace stored as binary object (BLOB). The record set can also contain the *Depth* and one field for each data point of the sonic trace. If no trace increment is specified the default value stored in the WellCAD.ini file will be taken.

Example:

```
. . .
# load a single Full Waveform Sonic Log
$logdata = SELECT * FROM $vers.datatable
LOADFULLWAVELOG('CBL',$logdata,4)
. . .

-OR-
. . .
# load all FWS Logs
LOOPON($loginfo)
     $logdata = SELECT Depth, Data
                FROM $loginfo.logtablename
     LOADFULLWAVELOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreFullWaveLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified FWS Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth (float)* and *Data (longbinary)*.
Additional values can be added to existing fields using the syntax *fieldname = value*. The *Logs $this* table contains a parameter *Increment* that can be used to store the trace increment for each FWS log.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (Depth float, Data longbinary)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STOREFULLWAVELOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadVspLog (*Log Title (text)* , *Log Data (record set)*)

LoadVspLog (*Log Data (record set)*)

Creates a new VSP Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Mud Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain at least a *Depth* and a data column. *Depth* must be the first field and should contain sorted depth values. All remaining fields are taken as data columns, with each column corresponding to one sample of the VSP data trace.

Example:

```
. . .
# load a single VSP Log
$logdata = SELECT * FROM $vers.datatable
LOADVSPLOG('VSP',$logdata)
. . .

-OR-
. . .
# load all VSP Logs
LOOPON($loginfo)
     $logdata = SELECT Depth, Data
                FROM $loginfo.logtablename
     LOADVSPLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreVspLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified VSP Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth* and at least one data column.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (Depth float, Col1 float, Col2 float,
+ Col3 float, Col4 float)

# store in it
$log = SELECT * FROM $logtablename
STOREVSPLOG($logs.Name, $log)

# done
CLOSE($log)
. . .
```

LoadDepthLog (*Log Title (text)* , *Log Data (record set)*)

LoadDepthLog (*Log Data (record set)*)

Creates one or more new Depth Logs in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Depth Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain at least a *Depth (float)* and a data column. *Depth* must be the first field and should contain sorted depth values. The second field is taken as data column. Each additional field is interpreted as data column as well.
If no log title is passed as first parameter or if more than one data column exists the field names are taken as log titles.

Example:

```
. . .
# load a single Depth Log
$LOGDATA = SELECT * FROM $vers.datatable
LOADDEPTHLOG('TVD',$logdata)
. . .

-OR-
. . .
# load all Depth Logs
LOOPON($loginfo)
     $logdata = SELECT Depth, Data
                FROM $loginfo.logtablename
     LOADDEPTHLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreDepthLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Depth Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth (float)* and *Data (float)* (or *Value* or the exact log title).
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (Depth float, Data float)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STOREDEPTHLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadOleLog (*Log Title (text)* , *Log Data (record set)*)

LoadOleLog (*Log Data (record set)*)

Creates one or more new Mud Logs in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the OLE Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain at least a *TopDepth (float)*, *BottomDepth (float)* and a data column (longbinary). All fields in the Log Data record set other than the depth information will be read as data stored as binary object (BLOB).

Example:

```
. . .
# load a single OLE Log
$LOGDATA = SELECT * FROM $vers.datatable
LOADOLELOG('Excel',$logdata)
. . .

-OR-
. . .
# load all OLE Logs
LOOPON($loginfo)
     $logdata = SELECT TopDepth, BottomDepth, OleItem
                FROM $loginfo.logtablename
     LOADOLELOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreOleLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified OLE Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *TopDepth (float), BottomDepth (float)* and *OleItem (longbinary)*.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (TopDepth float, BottomDepth float,
+ OleItem longbinary)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STOREOLELOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadAnalysisLog (*Log Title (text)* , *Log Data (record set)* , *Dictionary (record set)*)
LoadAnalysisLog (*Log Title (text)* , *Log Data (record set)* , *Dictionary (Text)*)
LoadAnalysisLog (*Log Title (text)* , *Log Data (record set)*)
LoadAnalysisLog (*Log Data (record set)*)

Creates a new Analysis Logs in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Analysis Log
Log Data	The record set containing the data
Dictionary	A record set or the path to a file containing the lithology patterns for the components.

Remarks:

The *Log Data* record set must contain at least a *Depth (float)* and a data column *(float)*. *Depth* must be the first field and should contain sorted depth values of constant sampling rate. All fields of the Log Data record set other than the *Depth* will be read as components of the Analysis Log.

Example:

```
. . .
# load a single Analysis Log
$logdata = SELECT * FROM $vers.datatable
LOADANALYSISLOG('Samples',$logdata,'C:\TEMP\Compos.lth)
. . .
-OR-
. . .
# load all Analysis Logs
LOOPON($loginfo)
     $logdata = SELECT Depth, Lime, Sand, Water
                FROM $loginfo.logtablename
     LOADANALYSISLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreAnalysisLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Analysis Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth (float)* and the fields corresponding to the component names used in the Analysis log. The data fields should be of format *(float).*
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (Depth float, Lime float, Sand float,
+ Water float)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STOREANALYSISLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadPercentageLog (*Log Title (text)* , *Log Data (record set)* , *Dictionary (record set)*)
LoadPercentageLog (*Log Title (text)* , *Log Data (record set)* , *Dictionary (Text)*)
LoadPercentageLog (*Log Title (text)* , *Log Data (record set)*)
LoadPercentageLog (*Log Data (record set)*)

Creates a new Percentage Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Percentage Log
Log Data	The record set containing the data
Dictionary	A record set or the path to a file containing the lithology patterns for the components.

Remarks:

The *Log Data* record set must contain at least a *Depth (float)* and a data column *(float)*. *Depth* must be the first field and should contain sorted depth values. All fields of the Log Data record set other than the *Depth* will be read as components of the Percentage Log.

Example:

```
. . .
# load a single Percentage Log
$logdata = SELECT * FROM $vers.datatable
LOADPERCENTAGELOG('Samples',$logdata,'C:\TEMP\Compos.lth)
. . .
-OR-
. . .
# load all Percentage Logs
LOOPON($loginfo)
     $logdata = SELECT Depth, Lime, Sand, Water
                FROM $loginfo.logtablename
     LOADPERCENTAGELOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StorePercentageLog (*Log Name (text)* , *Data Table (record set), Add Fields*)

Stores the data of the specified Percentage Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth (float)* and the fields corresponding to the component names used in the Percentage log. The data fields should be of type *(float).*
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (Depth float, Lime float, Sand float,
+ Water float)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STOREPERCENTAGELOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadBioLog (*Log Title (text)* , *Log Data (record set)* , *Dictionary (text)*)
LoadBioLog (*Log Title (text)* , *Log Data (record set)* , *Dictionary (record set)*)
LoadBioLog (*Log Data (record set)* , *Dictionary (text)*)
LoadBioLog (*Log Data (record set)* , *Dictionary (record set)*)

Creates a new Biostratigraphic Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Bio Log
Log Data	The record set containing the data
Dictionary	A record set or the path to a file containing the lithology patterns.

Remarks:

The *Log Data* record set must contain a *TopDepth (float) and BottomDepth (float)* column. Optional fields are *TaxonName (text)* and *Nature (text).* If the field *TaxonName* is defined the fields *SampleID (text)* and *Frequency (float)* are required, because the *LogData* record set is supposed to have one record per Bio item in this case. The fields *Symbol (text)* and *Comment (text)* are optional in this case. If the *TaxonName* field is not defined the *LogData* record set is supposed to have one record by sample.

Example:

```
$mylog = SELECT SampleID, TopDepth, BottomDepth, Nature, TaxonName
+ Frequency, Symbol, Comment
+ FROM $datatable
LOADBIOLOG($mylog)

-OR-

$mylog = SELECT TopDepth, BottomDepth, Grainstone, Algae
+ FROM $datatable
LOADBIOLOG($mylog)
```

StoreBioLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Bio Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *TopDepth (float) and BottomDepth (float)*. Optional fields are *TaxonName (text)* and *Nature (text).* If the field *TaxonName* is defined the field *Frequency (float)* is required, because the *LogData* record set is supposed to have one record per Bio item in this case. The fields *Symbol (text)* and *Comment (text)* are optional in this case. If the *TaxonName* field is not defined the *LogData* record set is supposed to have one record by sample.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
$mylog = SELECT TopDepth, BottomDepth, Nature, TaxonName
+ Frequency, Symbol, Comment
+ FROM $datatable
STOREBIOLOG('Bio',$mylog)

-OR-

$mylog = SELECT TopDepth, BottomDepth, Grainstone, Algae
+ FROM $datatable
STOREBIOLOG('Bio',$mylog)
```

LoadEngineeringLog (*Log Title (text)* , *Log Data (record set)*)

LoadEngineeringLog (*Log Data (record set)*)

Creates a new Engineering Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Engineering Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain the fields *Item (text)*, *TopDepth (float)*, *BottomDepth (float)*, *Position (float)*, *ExternalDiameter (float)*, *InternalDiameter (float)* and *InjectionDepth (float)*.

Example:

```
. . .
# load a single Percentage Log
$logdata = SELECT * FROM $vers.datatable
LOADENGINEERINGLOG('Borehole',$logdata)
. . .
-OR-
. . .
# load all Engineering Logs
LOOPON($loginfo)
     $logdata = SELECT *
                FROM $loginfo.logtablename
     LOADENGINEERINGLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreEngineeringLog (*Log Name (text)* , *Data Table (record set), Add Fields*)

Stores the data of the specified Engineering Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Item (text), TopDepth (float)*, *BottomDepth (float)*, *Position (float)*, *ExternalDiameter (float)*, *InternalDiameter (float)*, *InjectionDepth (float)*.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (Item varchar(255), TopDepth float
+ BottomDepth float, Position float, ExternalDiameter float,
+ InternalDiameter float)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STOREENGINEERINGLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadStackingPatternLog (*Log Title (text)* , *Log Data (record set)*)

LoadStackingPatternLog (*Log Data (record set)*)

Creates a new Stacking Pattern Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Stacking Pattern Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain a *TopDepth (float)*, *BottomDepth (float)*, *TopWidth (float)*, *BottomWidth (float)* column.

Example:

```
. . .
# load a single Stacking Pattern Log
$logdata = SELECT * FROM $vers.datatable
LOADSTACKINGPATTERNLOG('Trend',$logdata)
. . .
-OR-
. . .
# load all Logs
LOOPON($loginfo)
     $logdata = SELECT *
                FROM $loginfo.logtablename
     LOADSTACKINGPATTERNLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreStackingPatternLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Stacking Pattern Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *TopDepth (float)*, *BottomDepth (float)*, *TopWidth (float)*, *BottomWidth (float)*. Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (TopDepth float, BottomDepth float,
+ TopWidth float, BottomWidth float)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STORESTACKINGPATTERNLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadIntervalLog (*Log Title (text)* , *Log Data (record set)*)

LoadIntervalLog (*Log Data (record set)*)

Creates one or more new Interval Logs in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Interval Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain at least a *TopDepth (float)*, *BottomDepth (float)* and a data column. Each field contained in the *Log Data* record set other than the depth information will be imported as Interval Log using the field names as log titles.

Example:

```
. . .
# load a single Interval Log
$logdata = SELECT * FROM $vers.datatable
LOADINTERVALLOG('Test',$logdata)
. . .
-OR-
. . .
# load all Logs
LOOPON($loginfo)
     $logdata = SELECT *
                FROM $loginfo.logtablename
     LOADINTERVALLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreIntervalLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Interval Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *TopDepth (float)*, *BottomDepth (float)* and *Data (float)*.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (TopDepth float, BottomDepth float,
+ Data float)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STOREINTERVALLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadCoreDescLog (*Log Title (text)* , *Log Data (record set)*)

LoadCoreDescLog (*Log Data (record set)*)

Creates a new CoreDesc Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the CoreDesc Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain a *TopDepth (float)*, *BottomDepth (float)* and *Code (text)* field. The fields *Abundance (float)*, *Dominance (int)* and *Position (float)* are optional.

Example:

```
. . .
# load a single Interval Log
$logdata = SELECT * FROM $vers.datatable
LOADCOREDESCLOG('Fossils',$logdata)
. . .
-OR-
. . .
# load all Logs
LOOPON($loginfo)
     $logdata = SELECT *
                FROM $loginfo.logtablename
     LOADCOREDESCLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreCoreDescLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified CoreDesc Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *TopDepth (float)*, *BottomDepth (float)* and *Code (text)* field. The fields *Abundance (float)*, *Dominance (int)* and *Position (float)* are optional.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (TopDepth float, BottomDepth float,
+ Code varchar(255), Abundance float, Dominance int)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STORECOREDESCLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadStrataLog (*Log Title (text)* , *Log Data (record set)*)

LoadStrataLog (*Log Data (record set)*)

Creates a new Strata Log in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Strata Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain a *TopDepth (float)* and *BottomDepth (float)* field. Optional fields are *TopContact (text)* and *BottomContact (text)*. One strata log column is created for each additional field contained in the *LogData* record set. The field names will be used as strata log column titles.

Example:

```
. . .
# load a single Strata Log
$logdata = SELECT * FROM $vers.datatable
LOADSTRATALOG('Chrono Strat',$logdata)
. . .
-OR-
. . .
# load all Logs
LOOPON($loginfo)
     $logdata = SELECT *
                FROM $loginfo.logtablename
     LOADSTRATALOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreStrataLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data of the specified Strata Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *TopDepth (float)* and *BottomDepth (float)*. The fields *TopContact (text)* and *BottomContact (text)* are optional. For each strata log column a field should exist with its name matching the strata log column title.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (TopDepth float, BottomDepth float,
+ System varchar(255), Series varchar(255), Stage varchar(255))

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STORESTRATALOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadCrossLog (*Log Title (text)* , *Log Data (record set)*)

LoadCrossLog (*Log Data (record set)*)

Creates new Cross Logs in WellCAD and downloads the log data from the database.

Parameter:

Log Title	The title of the Cross Section Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain either a *TopDepth (float)* and *BottomDepth (float)* column or at least one column with depth values. If no *TopDepth* and *BottomDepth* fields are defined, each field of the *LogData* record set will be imported as a new Cross Section log. As the Cross Section log does not contain data only the top and bottom depth information of each cross section box is imported.

Example:

```
. . .
# load a single Cross Section Log
$logdata = SELECT * FROM $vers.datatable
LOADCROSSLOG('XSection',$logdata)
. . .
-OR-
. . .
# load all Logs
LOOPON($loginfo)
     $logdata = SELECT *
                FROM $loginfo.logtablename
     LOADCROSSLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreCrossLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data (depth intervals) of the specified Cross Section Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *Depth (float)* or *TopDepth (float)* and *BottomDepth (float)*.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (TopDepth float, BottomDepth float)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STORECROSSLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadPolarLog (*Log Title (text)* , *Log Data (record set)*)

LoadPolarLog (*Log Data (record set)*)

Creates a new Polar & Rose Log in WellCAD and downloads the log data (depth intervals) from the database.

Parameter:

Log Title	The title of the Polar & Rose Log
Log Data	The record set containing the data

Remarks:

The *Log Data* record set must contain at a *TopDepth (float)* and *BottomDepth (float)* column. *Description (text)* is an optional field. Only the depth position of each box is loaded and (optional) the description text. The Polar & Rose log does not contain its own data.

Example:

```
. . .
# load a single Polar & Rose Log
$logdata = SELECT * FROM $vers.datatable
LOADPOLARLOG('Statistic',$logdata)
. . .
-OR-
. . .
# load all Logs
LOOPON($loginfo)
     $logdata = SELECT *
                FROM $loginfo.logtablename
     LOADPOLARLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StorePolarLog (*Log Name (text)* , *Data Table (record set), Add Fields...*)

Stores the data (depth intervals) of the specified Polar & Rose Log to the database.

Parameter:

Log Name	The name of the log to be stored.
Data Table	A record set containing the fields necessary to store the log data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the log data. The fields required are: *TopDepth (float)* and *BottomDepth (float)*. *Description (text)* is an optional field. Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (TopDepth float, BottomDepth float,
+ Description longtext)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STOREPOLARLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

LoadAnnotationLog (*Layer Title (text)* , *Layer Data (record set)*)

LoadAnnotationLog (*Layer Data (record set)*)

Creates a new Annotation Layer in WellCAD and downloads the layer contents from the database.

Parameter:

Layer Title	The title of the Annotation layer
Layer Data	The record set containing the data

Remarks:

The *Log Data* record set must contain a *Dummy (int)* and *Data (longbinary)* field. The *Dummy* field is not read but must be contained for compatibility reasons. The function loads the entire contents of the Annotation layer, which is stored within the binary object (BLOB).

Example:

```
. . .
# load a single Annotation Log
$logdata = SELECT * FROM $vers.datatable
LOADANNOTATIONLOG('Comments',$logdata)
. . .
-OR-
. . .
# load all Logs
LOOPON($loginfo)
     $logdata = SELECT *
                FROM $loginfo.logtablename
     LOADANNOTATIONLOG($logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
. . .
```

StoreAnnoationLog (*Layer Name (text)* , *Data Table (record set), Add Fields...*)

Stores the entire contents of the specified Annotation Layer to the database.

Parameter:

Layer Name	The name of the layer to be stored.
Data Table	A record set containing the fields necessary to store the layer data.
Add Fields	Additional data can be assigned to fields of the *Data Table* record set.

Remarks:

The *Data Table* record set must contain all fields necessary to store the layer data. The fields required are: *Dummy (int)* and *Data (longbinary)*.
Additional values can be added to existing fields using the syntax *fieldname = value*.

Example:

```
. . .
# create the LogTable
CREATE TABLE $logtablename (Dummy int, Data longbinary)

# add additional columns for additional fields
ALTER TABLE $logtablename
+ ADD RunNumber varchar(255), HoleID varchar(255)

# store in it
$log = SELECT * FROM $logtablename
STOREANNOTATIONLOG($logs.Name, $log, RunNumber = $run, HoleID = &uwi)

# done
CLOSE($log)
. . .
```

CreatePlugTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the plug symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the plug symbol:
plugID (number), plugNumber (text), plugTop (number), plugBottom (number), plugType (text).

The *plugType* value will be set to “Bridge” or “Cement” when storing data to this table. To be recognized the *plugType* value must have been set to “Bridge” or “Cement” when loading data from this table into WellCAD.

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$PlugDataTableName = wcPLUG$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET PlugDataTable = '$PlugDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Plug Table and store
CREATEPLUGTABLE($PlugDataTableName)
$layer = SELECT * FROM $PlugDataTableName
STOREPLUGS($Layers.Name,$layer)
CLOSE($layer)
```

StorePlugs (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all plug symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

The *Data Table* record set must contain the following field names:
plugID (number), plugNumber (text), plugTop (number), plugBottom (number), plugType (text).
A call of the *CreatePlugTable* function prior to the execution of the *StorePlugs* function generates the correct table structure. Display settings are not saved with this function. A set of default display options is saved with the document template.

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$PlugDataTable = wcPLUG$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET PlugDataTable = '$PlugDataTable'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Plug Table and store
CREATEPLUGTABLE($PlugDataTableName)
$data = SELECT * FROM $PlugDataTableName
STOREPLUGS($Layers.Name,$data)
CLOSE($data)
```

LoadPlugs (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the plug data.
Layer Name	The name of the annotation layer to display the plug symbol in.

Remarks:

The *Data* Table record set must contain the following fields:
plugID (number), plugNumber (text), plugTop (number), plugBottom (number), plugType (text).
The *plugType* value should be set to “Bridge” or “Cement” to differentiate the two symbols. A set of default display settings like horizontal offset, color etc. is applied when loading a document template file.

Example:

```
# load plug data
$Plugdata = SELECT * FROM $Layers.PlugDataTable
LoadPlugs($Plugdata,'$Layers.Title')
CLOSE($Plugdata)
```

CreateCasingTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the casing symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the plug symbol:
casingID (number), casingDepth (number), casingSize (text), casingType (text).

The *casingType* value will be set to “Liner” or “Shoe” when storing data to this table. To be recognized the *casingType* value must have been set to “Liner” or “Shoe” when loading data from this table into WellCAD.

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$CasingDataTableName = wcCASING$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET CasingDataTable = '$CasingDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Casing Table and store
CREATECASINGTABLE($CasingDataTableName)
$data = SELECT * FROM $CasingDataTableName
STORECASINGINFO($Layers.Name,$data)
CLOSE($data)
```

StoreCasingInfo (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all casing symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

The *Data Table* record set must contain the following field names:
casingID (number), casingDepth (number), casingSize (text), casingType (text).
A call of the *CreateCasingTable* function prior to the execution of *StoreCasingInfo* generates the correct table structure. Display settings are not saved with this function. A set of default display options is saved with the document template.

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$CasingDataTable = wcCASING$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET CasingDataTable = '$CasingDataTable'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Casing Table and store
CREATECASINGTABLE($CasingDataTableName)
$data = SELECT * FROM $CasingDataTableName
STORECASINGINFO($Layers.Name,$data)
CLOSE($data)
```

LoadCasingInfo (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the casing symbol in.

Remarks:

The *Data* Table record set must contain the following fields:
casingID (number), casingDepth (number), casingSize (text), casingType (text).
The *casingType* value should be set to “Liner” or “Shoe” to differentiate the two symbols. A set of default display settings like horizontal offset, color etc. is applied when loading a document template file.

Example:

```
# load casing data
$Casingdata = SELECT * FROM $Layers.CasingDataTable
LOADCASINGINFO($Casingdata,'$Layers.Title')
CLOSE($Casingdata)
```

CreateShowsTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the show symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the shows symbol:
showID (number), showTop (number), showBottom (number), showTransTop (number), showType (text), showQuality (text)

The *showType* value will be set to “Oil”, “Gas”, “Oil&Gas”, “Fluorescence”, “Tarry Oil” or “Dead Oil” when storing data to this table. The *showQuality* value will be: “Trace”, “Poor”, “Moderate”, “Good” or “Excellent”.

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$ShowsDataTableName = wcSHOWS$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET ShowsDataTable = '$ShowsDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Shows Table and store
CREATESHOWSTABLE($ShowsDataTableName)
$data = SELECT * FROM $ShowsDataTableName
STORESHOWS($Layers.Name,$data)
CLOSE($data)
```

StoreShows (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all shows symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$ShowsDataTableName = wcSHOWS$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET ShowsDataTable = '$ShowsDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Shows Table and store
CREATESHOWSTABLE($ShowsDataTableName)
$data = SELECT * FROM $ShowsDataTableName
STORESHOWS($Layers.Name,$data)
CLOSE($data)
```

LoadShows (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the shows symbol in.

Remarks:

Example:

```
# load shows data
$data = SELECT * FROM $Layers.ShowsDataTable
LOADSHOWS($data,'$Layers.Title')
CLOSE($data)
```

CreateConvCoreTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the conventional core symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the conventional core symbol:
coreID (number), coreNumber (text), drillerTop (number), drillerBottom (number), loggerTop (number), coreRecovery (number)

The *coreRecovery* value will be set as percentage between 0.0 and 100.0.

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$ConvCoreDataTableName = wcCONVCORE$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET ConvCoreDataTable = '$ConvCoreDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Conventional Cores Table and store
CREATECONVCORETABLE($ConvCoreDataTableName)
$data = SELECT * FROM $ConvCoreDataTableName
STORECONVCORES($Layers.Name,$data)
CLOSE($data)
```

StoreConvCores (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all conventional core symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$ConvCoreDataTableName = wcCONVCORE$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET ConvCoreDataTable = '$ConvCoreDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Conventional Cores Table and store
CREATECONVCORETABLE($ConvCoreDataTableName)
$data = SELECT * FROM $ConvCoreDataTableName
STORECONVCORES($Layers.Name,$data)
CLOSE($data)
```

LoadConvCores (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the conventional core symbol in.

Remarks:

Example:

```
# load conventional core data
$data = SELECT * FROM $Layers.ConvCoreDataTable
LOADCONVCORES($data,'$Layers.Title')
CLOSE($data)
```

CreateSWCTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the sidewall core symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the sidewall core symbol:
swcID (number), swcDepth (number), swcLithology (text), swcRunNumber (text), swcShotNumber (text), swcRecovery (number)

The *swcRecovery* value will be set to 0.0 if the “Not Recovered” flag was set in WellCAD. If the flag was set to “Recovered” a value of 100.0 will be set when data is stored.

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$SWCDataTableName = wcSWC$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET SWCDataTable = '$SWCDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the SWC Table and store
CREATESWCTABLE($SWCDataTableName)
$data = SELECT * FROM $SWCDataTableName
STORESIDECORES($Layers.Name,$data)
CLOSE($data)
```

StoreSideCores (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all sidewall core symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$SWCDataTableName = wcSWC$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET SWCDataTable = '$SWCDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the SWC Table and store
CREATESWCTABLE($SWCDataTableName)
$data = SELECT * FROM $SWCDataTableName
STORESIDECORES($Layers.Name,$data)
CLOSE($data)
```

LoadSideCores (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the sidewall core symbol in.

Remarks:

Example:

```
# load sidewall core data
$data = SELECT * FROM $Layers.SWCDataTable
LOADSIDECORES($data,'$Layers.Title')
CLOSE($data)
```

CreateDeviationTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the deviation/orientation symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the deviation symbol:
surveyID (number), surveyDepth (number), surveyInclination (number), surveyAzimuth (number)

The *surveyInclination* and *surveyAzimuth* values are in degree.

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$DeviationDataTableName = wcDeviation$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET DeviationDataTable = '$DeviationDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Deviation Table and store
CREATEDEVIATIONTABLE($DeviationDataTableName)
$data = SELECT * FROM $DeviationDataTableName
STOREDEVIATIONINFO($Layers.Name,$data)
CLOSE($data)
```

StoreDeviationInfo (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all deviation/orientation symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$DeviationDataTableName = wcDeviation$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET DeviationDataTable = '$DeviationDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Deviation Table and store
CREATEDEVIATIONTABLE($DeviationDataTableName)
$data = SELECT * FROM $DeviationDataTableName
STOREDEVIATIONINFO($Layers.Name,$data)
CLOSE($data)
```

LoadDeviationInfo (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the deviation/orientation symbol in.

Remarks:

Example:

```
# load deviation symbol data
$data = SELECT * FROM $Layers.DeviationDataTable
LOADDEVIATIONINFO($data,'$Layers.Title')
CLOSE($data)
```

CreateRftTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the RFT symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the RFT symbol:
rftID (number), rftDepth (number), rftRun (text), rftTest (text), rftPressure (number), rftPressUnit (text), rftTestRating (text), rftFluidSample (text)

The *rftTestRating* value can be set to “Default”, “Poor”, “Fair”, “Good”, “Excellent”, “Tight”, “Supercharged”, “Lost Seal” or “Failed”.
The *rftFluidSample* value can be set to “yes” or “no”.

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$RFTDataTableName = wcRFT$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET RFTDataTable = '$RFTDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the RFT Table and store
CREATERFTTABLE($RFTDataTableName)
$data = SELECT * FROM $RFTDataTableName
STORERFTS($Layers.Name,$data)
CLOSE($data)
```

StoreRFTs (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all RFT symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$RFTDataTableName = wcRFT$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET RFTDataTable = '$RFTDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the RFT Table and store
CREATERFTTABLE($RFTDataTableName)
$data = SELECT * FROM $RFTDataTableName
STORERFTS($Layers.Name,$data)
CLOSE($data)
```

LoadRFTs (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the RFT symbol in.

Remarks:

Example:

```
# load RFT symbol data
$data = SELECT * FROM $Layers.RFTDataTable
LOADRFTS($data,'$Layers.Title')
CLOSE($data)
```

CreateMudTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the Mud Info symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the Mud Info symbol:
fluidID (number), fluidDepth (number), fluidType (text), fluidDensity (number), fluidDensUnit (text), fluidOilContent (number), fluidOilUnit (text), fluidFunnelViscosity (number), fluidFunnelUnit (text), fluidPlasticViscosity (number), fluidPlasticUnit (text), fluidYieldPoint (number), fluidYieldUnit (text), fluidGelsMinutes (number), fluidGelsUnitM (text), fluidGelsSeconds (number), fluidGelsUnitS (text)

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$MudDataTableName = wcMud$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET MudDataTable = '$MudDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Mud Table and store
CREATEMUDTABLE($MudDataTableName)
$data = SELECT * FROM $MudDataTableName
STOREMUDINFO($Layers.Name,$data)
CLOSE($data)
```

StoreMudInfo (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all Mud Info symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$MudDataTableName = wcMud$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET MudDataTable = '$MudDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Mud Table and store
CREATEMUDTABLE($MudDataTableName)
$data = SELECT * FROM $MudDataTableName
STOREMUDINFO($Layers.Name,$data)
CLOSE($data)
```

LoadMudInfo (*Data Table (record set)*, *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the Mud Info symbol in.

Remarks:

Example:

```
# load Mud Info symbol data
$data = SELECT * FROM $Layers.MudDataTable
LOADMUDINFO($data,'$Layers.Title')
CLOSE($data)
```

CreateBitTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the Bit Info symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the Bit Info symbol:
bitID (number), bitDepth (number), bitNumber (text), bitSize (text), bitType (text), bitDepIn (number), bitDepOut (number), bitDistance (number), bitHours (number), IADCInner (number), IADCOuter (number), IADCDull (text), IADCLocation (text), IADCBearing (text), IADCGauge (text), IADCOther (text), IADCReason (text)

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$BitDataTableName = wcBit$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET BitDataTable = '$BitDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the RFT Table and store
CREATEBITTABLE($BitDataTableName)
$data = SELECT * FROM $BitDataTableName
STOREBITINFO($Layers.Name,$data)
CLOSE($data)
```

StoreBitInfo (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all Bit Info symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$BitDataTableName = wcBit$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET BitDataTable = '$BitDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the RFT Table and store
CREATEBITTABLE($BitDataTableName)
$data = SELECT * FROM $BitDataTableName
STOREBITINFO($Layers.Name,$data)
CLOSE($data)
```

LoadBitInfo (*Data Table (record set)*, *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the Bit Info symbol in.

Remarks:

Example:

```
# load Bit Info symbol data
$data = SELECT * FROM $Layers.BitDataTable
LOADBITINFO($data,'$Layers.Title')
CLOSE($data)
```

CreateDstTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the DST symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the DST symbol:
dstID (number), dstNumber (text), dstTop (number), dstBottom (number), dstIntTop1 (number), dstIntBot1 (number), dstIntTop2 (number), dstIntBot2 (number), dstIntTop3 (number), dstIntBot3 (number), dstIntTop4 (number), dstIntBot4 (number), dstIntTop5 (number), dstIntBot5 (number), dstIntTop6 (number), dstIntBot6 (number), dstIntTop7 (number), dstIntBot7 (number), dstIntTop8 (number), dstIntBot8 (number), dstIntTop9 (number), dstIntBot9 (number), dstIntTop10(number), dstIntBot10 (number)

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$DSTDataTableName = wcDST$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET DSTDataTable = '$DSTDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the DST Table and store
CREATEDSTTABLE($DSTDataTableName)
$data = SELECT * FROM $DSTDataTableName
STOREDSTS($Layers.Name,$data)
CLOSE($data)
```

StoreDSTs (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all DST symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$DSTDataTableName = wcDST$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET DSTDataTable = '$DSTDataTableName'
+ WHERE LayerID = $layer.LayerID
A($layer)

# create the DST Table and store
CREATEDSTTABLE($DSTDataTableName)
$data = SELECT * FROM $DSTDataTableName
STOREDSTS($Layers.Name,$data)
CLOSE($data)
```

LoadDSTs (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the DST symbol in.

Remarks:

Example:

```
# load DST symbol data
$data = SELECT * FROM $Layers.DSTDataTable
LOADDSTS($data,'$Layers.Title')
CLOSE($data)
```

CreateProdLinerTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the Production Liner symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the Production Liner symbol:
linerID (number), linerTop (number), linerBottom (number), linerIntTop1 (number), linerIntBot1 (number), linerIntTop2 (number), linerIntBot2 (number), linerIntTop3 (number), linerIntBot3 (number), linerIntTop4 (number), linerIntBot4 (number), linerIntTop5 (number), linerIntBot5 (number), linerIntTop6 (number), linerIntBot6 (number), linerIntTop7 (number), linerIntBot7 (number), linerIntTop8 (number), linerIntBot8 (number), linerIntTop9 (number), linerIntBot9 (number), linerIntTop10(number), linerIntBot10 (number)

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$ProdLinerDataTableName = wcProdLiner$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET ProdLinerDataTable = '$ProdLinerDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Production Liner Table and store
CREATEPRODLINERTABLE($ProdLinerDataTableName)
$data = SELECT * FROM $ProdLinerDataTableName
STOREPRODLINER ($Layers.Name,$data)
CLOSE($data)
```

StoreProdLiner (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all Production Liner symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$ProdLinerDataTableName = wcProdLiner$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET ProdLinerDataTable = '$ProdLinerDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Production Liner Table and store
CREATEPRODLINERTABLE($ProdLinerDataTableName)
$data = SELECT * FROM $ProdLinerDataTableName
STOREPRODLINER ($Layers.Name,$data)
CLOSE($data)
```

LoadProdLiner (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the Production Liner symbol in.

Remarks:

Example:

```
# load Production Liner symbol data
$data = SELECT * FROM $Layers.ProdLinerDataTable
LOADPRODLINER($data,'$Layers.Title')
CLOSE($data)
```

CreateDateMarkerTable (*Table Name (text)*)

Creates a new database table and inserts the fields to hold the data of the Date Marker symbols.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the Date Marker symbol:
markerID (number), markerDepth (number), markerText (text), markerYear (number), markerMonth (number), markerDay (number), markerHour (number), markerMinute (number), markerSecond (number)

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$DateDataTableName = wcDate$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET DateDataTable = '$DateDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Date Marker Table and store
CREATEDATEMARKERTABLE($DateDataTableName)
$data = SELECT * FROM $DateDataTableName
STOREDATEMARKER($Layers.Name,$data)
CLOSE($data)
```

StoreDateMarker (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all Date Marker symbols of the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$DateDataTableName = wcDate$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET DateDataTable = '$DateDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Date Marker Table and store
CREATEDATEMARKERTABLE($DateDataTableName)
$data = SELECT * FROM $DateDataTableName
STOREDATEMARKER($Layers.Name,$data)
CLOSE($data)
```

LoadDateMarker (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the Date Marker symbol in.

Remarks:

Example:

```
# load Date Marker symbol data
$data = SELECT * FROM $Layers.DateMarkerDataTable
LOADDATEMARKER($data,'$Layers.Title')
CLOSE($data)
```

CreateStringAnnotationTable (*Table Name (text)*)

Creates a new database table and inserts the fields to store the data of the standard String Annotation.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the String Annotation symbol:
annoTop (number), annoBottom (number), annoLeft (number), annoRight (number), annoText (text)

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$StringDataTableName = wcString$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET StringDataTable = '$StringDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the String Annotation Table and store
CREATESTRINGANNOTATIONTABLE($StringDataTableName)
$data = SELECT * FROM $StringDataTableName
STORESTRINGANNOTATION($Layers.Name,$data)
CLOSE($data)
```

StoreStringAnnotation (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all String Annotations from the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$StringDataTableName = wcString$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET StringDataTable = '$StringDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the String Annotation Table and store
CREATESTRINGANNOTATIONTABLE($StringDataTableName)
$data = SELECT * FROM $StringDataTableName
STORESTRINGANNOTATION($Layers.Name,$data)
CLOSE($data)
```

LoadStringAnnotation (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the String Annotation symbol in.

Remarks:

Example:

```
# load String Annotation data
$data = SELECT * FROM $Layers.StringAnnotationDataTable
LOADSTRINGANNOTATION($data,'$Layers.Title')
CLOSE($data)
```

CreateBitmapAnnotationTable (*Table Name (text)*)

Creates a new database table and inserts the fields to store the data of the standard Bitmap Annotation.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the Bitmap Annotation symbol:
annoTop (number), annoBottom (number), annoLeft (number), annoRight (number), annoBitmap (long binary)

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$BitmapDataTableName = wcBitmap$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET BitmapDataTable = '$BitmapDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Bitmap Annotation Table and store
CREATEBITMAPANNOTATIONTABLE($BitmapDataTableName)
$data = SELECT * FROM $BitmapDataTableName
STOREBITMAPANNOTATION($Layers.Name,$data)
CLOSE($data)
```

StoreBitmapAnnotation (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all Bitmap Annotations from the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$BitmapDataTableName = wcBitmap$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET BitmapDataTable = '$BitmapDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Bitmap Annotation Table and store
CREATEBITMAPANNOTATIONTABLE($BitmapDataTableName)
$data = SELECT * FROM $BitmapDataTableName
STOREBITMAPANNOTATION($Layers.Name,$data)
CLOSE($data)
```

LoadBitmapAnnotation (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the Bitmap Annotation symbol in.

Remarks:

Example:

```
# load Bitmap Annotation data
$data = SELECT * FROM $Layers.BitmapAnnotationDataTable
LOADBITMAPANNOTATION($data,'$Layers.Title')
CLOSE($data)
```

CreateArrowAnnotationTable (*Table Name (text)*)

Creates a new database table and inserts the fields to store the data of the standard Arrow Annotation.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the Arrow Annotation symbol:
annoTop (number), annoBottom (number), annoLeft (number), annoRight (number)

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$ArrowDataTableName = wcArrow$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET ArrowDataTable = '$ArrowDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Arrow Annotation Table and store
CREATEARROWANNOTATIONTABLE($ArrowDataTableName)
$data = SELECT * FROM $ArrowDataTableName
STOREARROWANNOTATION($Layers.Name,$data)
CLOSE($data)
```

StoreArrowAnnotation (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all Arrow Annotations from the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$ArrowDataTableName = wcArrow$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET ArrowDataTable = '$ArrowDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Arrow Annotation Table and store
CREATEARROWANNOTATIONTABLE($ArrowDataTableName)
$data = SELECT * FROM $ArrowDataTableName
STOREARROWANNOTATION($Layers.Name,$data)
CLOSE($data)
```

LoadArrowAnnotation (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the Arrow Annotation symbol in.

Remarks:

Example:

```
# load String Annotation data
$data = SELECT * FROM $Layers.ArrowAnnotationDataTable
LOADARROWANNOTATION($data,'$Layers.Title')
CLOSE($data)
```

CreateStringCalloutTable (*Table Name (text)*)

Creates a new database table and inserts the fields to store the data of the standard String Callout.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the String Callout symbol:
annoTop (number), annoBottom (number), annoLeft (number), annoRight (number), arrowTop (number), arrowLeft (number), annoData (long binary)

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$StringCalloutTableName = wcStringCallout$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET StringCalloutDataTable = '$StringCalloutDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the String Callout Table and store
CREATESTRINGCALLOUTTABLE($StringCalloutDataTableName)
$data = SELECT * FROM $StringCalloutDataTableName
STORESTRINGCALLOUT($Layers.Name,$data)
CLOSE($data)
```

StoreStringCallout (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all String Callouts from the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID and Title = '$Layers.Name'
$StringCalloutTableName = wcStringCallout$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET StringCalloutDataTable = '$StringCalloutDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the String Callout Table and store
CREATESTRINGCALLOUTTABLE($StringCalloutDataTableName)
$data = SELECT * FROM $StringCalloutDataTableName
STORESTRINGCALLOUT($Layers.Name,$data)
CLOSE($data)
```

LoadStringCallout (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the String Callout symbol in.

Remarks:

Example:

```
# load String Annotation data
$data = SELECT * FROM $Layers.StringCalloutDataTable
LOADSTRINGCALLOUT($data,'$Layers.Title')
CLOSE($data)
```

CreateBitmapCalloutTable (*Table Name (text)*)

Creates a new database table and inserts the fields to store the data of the standard Bitmap Callout.

Parameter:

Table Name The name of the new table.

Remarks:

Executes the standard SQL command *CreateTable,* setting the name of the table. The following default fields are inserted for the Bitmap Callout symbol:
annoTop (number), annoBottom (number), annoLeft (number), annoRight (number), arrowTop (number), arrowLeft (number), annoData (long binary)

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$BitmapCalloutDataTableName = wcBitmapCallout$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET BitmapCalloutDataTable = '$BitmapCalloutDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Bitmap Callout Table and store
CREATEBITMAPCALLOUTTABLE($BitmapCalloutDataTableName)
$data = SELECT * FROM $BitmapCalloutDataTableName
STOREBITMAPCALLOUT($Layers.Name,$data)
CLOSE($data)
```

StoreBitmapCallout (*Layer Name (text)* , *Data Table (record set)*)

Stores the data of all Bitmap Callouts from the specified annotation layer to the data table.

Parameter:

Layer Name	The name identifies the annotation layer the data is taken from.
Data Table	A record set to store the data.

Remarks:

Example:

```
# get a unique DataTableName based on our unique LayerID
$layer = SELECT LayerID FROM Layer
+ WHERE VersionID = $VERSID AND Title = '$Layers.Name'
$BitmapCalloutDataTableName = wcBitmapCallout$layer.LayerID

# update the Layer row, insert the new table name
UPDATE Layer SET BitmapCalloutDataTable = '$BitmapCalloutDataTableName'
+ WHERE LayerID = $layer.LayerID
CLOSE($layer)

# create the Bitmap Callout Table and store
CREATEBITMAPCALLOUTTABLE($BitmapCalloutDataTableName)
$data = SELECT * FROM $BitmapCalloutDataTableName
STOREBITMAPCALLOUT($Layers.Name,$data)
CLOSE($data)
```

LoadBitmapCallout (*Data Table (record set)* , *Layer Name (text)*)

Inserts a new annotation layer of the specified name if necessary and loads all data from the *Data Table*.

Parameter:

Data Table	The record set containing the casing data.
Layer Name	The name of the annotation layer to display the Bitmap Callout symbol in.

Remarks:

Example:

```
# load String Annotation data
$data = SELECT * FROM $Layers.BitmapCalloutDataTable
LOADBITMAPCALLOUT($data,'$Layers.Title')
CLOSE($data)
```

OpenTemplate (*Template File Path (text)* , *bPrompt (bool)* , *bDocTemp (bool)* , *bCreateNewLogs (bool)* , *bCreate NewLayers (bool)* , *bApplyAnnoSettings (bool)*)
OpenTemplate (*TempRset (record set)* , *bPrompt (bool)* , *bDocTemp (bool)* , *bCreateNewLogs (bool)* , *bCreate NewLayers (bool)* , *bApplyAnnoSettings (bool)*)

Opens a template and applies it to the current WellCAD document.

Parameter:

Template File Path	The full path to the *.wdt or *.wct template file.
TempRset	A record set containing a template formerly saved with *SaveTemlate*.
bPrompt	If set to TRUE the user will be asked to select a log each time a log contained in the template could not be found in the current borehole document. The default is TRUE.
bDocTemplate	If set to TRUE it indicates that the template that will be loaded is a WellCAD Document (*.wdt) template. FALSE would indicate a WellCAD Layout (*.wct) template. The default is FALSE.
bCreateNewLogs	Indicates whether logs contained in the template and not in the current borehole document are created (TRUE) or not (FALSE). The default is FALSE.
bCreateNewlayers	Indicates whether annotation layers contained in the template and not in the current borehole document are created (TRUE) or not (FALSE). The default is FALSE.
bApplyAnnoSettings	Indicates whether the default display settings for annotations (operational symbols) stored for each layer are applied (TRUE) or not (FALSE). All symbols of the same annotation layer will get the same look.

Remarks:

If the template is stored within the database it should be stored as long text if it is layout template (*.wct) and stored as long binary object (BLOB) if it is a document template (*.wdt). The *bDocTemp* parameter should be set according the data format. (see *SaveTemplate*).

Example:

```
# load template
$Template = SELECT Template, VersionID FROM FileVersion
+ WHERE VersionID = $VERSID
OPENTEMPLATE($Template, FALSE, TRUE, TRUE, TRUE, TRUE)
CLOSE($Template)
```

SaveTemplate (*TempRset (record set)* , *bDocTemp (bool)*)

Stores either a WellCAD layout or document template to the database.

Parameter:

TempRset	The record set the template will be stored to.
bDocTemp	If set to TRUE a document template will be saved as long binary object (BLOB). A FALSE would initiate the saving of a layout template as long text.

Remarks:

The *bDocTemplate* should be chosen corresponding to the same parameter used in the *OpenTemplate* function. (see *OpenTemplate*).

Example:

```
# store template
$template = SELECT Template, VersionID FROM FileVersion
+ WHERE VersionID = $VERSID
SAVETEMPLATE($template, TRUE)
CLOSE($template )
```

EnableTrace (*bTrace (bool)*)

Enables or disables the display of the trace window.

Parameter:

bTrace	Can be TRUE or FALSE

Remarks:

If the *bTrace* parameter is set to TRUE the trace window is displayed while the SQL script is executed. Once the script execution has finished the user must close the window. The trace window shows all the command strings thrown out while the SQL script is executed.

Example:

```
ENABLETRACE(TRUE)
```

IgnoreError (*bIgnore (bool)*)

If set to TRUE it will cause the script to continue execution even if an SQL error occurs.

Parameter:

bIgnore Can be TRUE or FALSE

Remarks:

When *bIgnore* parameter is set to TRUE and errors are ignored, they are still displayed in the trace window. The script execution will not be aborted even if an SQL error occurs.

Example:

```
# delete all logs, an error might occur if the descID does not exist yet
# but it is skipped by the IgnoreError flag
IGNOREERROR(true)
$Logs = SELECT LogTableName FROM Log WHERE descID = $DESCID
LOOPON($Logs)
    # delete all logs
    DROP TABLE $Logs.LogTableName
ENDLOOP
IGNOREERROR(false)
```

LoopOn (*RSET (record set)*)

Executes the statement between *LoopOn* and *EndLoop* for all records in the passed *RSET* record set.

Parameter:

RSET	A record set containing all the records that should be looped on.

Remarks:

The function loops on all records contained in the *RSET* record set and opens each record to execute the statements within the loop. The *LoopOn* function should always be used in conjunction with the *EndLoop* command.

Example:

```
# load logs
$loginfo = SELECT * FROM Log WHERE VersionID = $versid
# and loop on it
LOOPON($loginfo)
     $logdata = SELECT * FROM $loginfo.LogTableName
     LOADLOG($loginfo,$logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
```

EndLoop

Ends a loop that has been initialized with the *LoopOn* command before. (see *LoopOn*)

Example:

```
# load logs
$loginfo = SELECT * FROM Log WHERE VersionID = $versid
# and loop on it
LOOPON($loginfo)
     $logdata = SELECT * FROM $loginfo.LogTableName
     LOADLOG($loginfo,$logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($loginfo)
```

ImportFile (*FilePath* (text))

Imports a data file if the file format is supported by WellCAD.

Parameter:

FilePath A text string defining the path and file name.

Remarks:

The file format is derived from the extension of the file name (e.g. test.csv).

Example:

```
# load a file
IMPORTFILE($myrecordset.AsciiFileName)

-OR-

IMPORTFILE('C:\Temp\Test.csv')
```

SingleSelectDlg (*RSET* (record set), *Field* (text), *Title* (text))

Displays a dialog box, which allows the user to select a single record from the record set.

Parameter:

RSET	The recordset containing the data that is displayed in the list box of the dialog.
Field	Specifies the fieldname whose value of the selected record is returned by the function.
Title	The string passed with this parameter is displayed as message within the dialog box.

Remarks:

The function displays a dialog box with a list showing the contents of the passed record set. The user can select a single record. The return value of the function is a value from the selected record set which is identified by the *Field* parameter. The *Field* name must be contatined in the *RSET*.

Example:

```
# single select to retrieve a specific WellID
$wells = SELECT UWI, Title, WellID
+ FROM Wells
+ WHERE FieldID = $fieldid
$wellid = SINGLESELECTDLG($wells, WellID, 'Select the well to load')
```

SinglePromptDlg (*Title (text)*)
SinglePromptDlg (*Title (text), Value (text)*)

Displays a dialog box that lets the user set the value of a variable.

Parameter:

Title	The text message used in the dialog box.
Value	The default value that is returned by the function if the user does not alter the value.

Remarks:

The function displays a dialog box with a single edit box. The user can type in a value which is returned by the function and assigned to a variable.

Example:

```
# prompt user for new project details
$name = SINGLEPROMPTDLG('Enter name of the new project')
$desc = SINGLEPROMPTDLG('Enter a description for the new project')
. . .
# insert new row into the project table
INSERT INTO projects (Name, Description)
+ VALUES ('$name', '$desc')
+ USING $dsn
```

MultiSelectDlg (*RSET* (record set), *Title* (text))

Displays a dialog box, which allows the user to select multiple records from the record set.

Parameter:

RSET	The recordset containing the data that is displayed in the list box of the dialog.
Title	The string passed with this parameter is displayed as message within the dialog box.

Remarks:

The function displays a dialog box with a list showing the contents of the passed record set. The user can select multiple records from that list. When the user clicks the OK button the *RSET* record set will be updated to contain only the selected records.

Example:

```
# fill the record set with all logs
$logs = SELECT Title, DataTable, LogType
+ FROM Logs
# multi select to get a record set of selected logs
MULTISELECTDLG($logs, 'Select the logs to load')
# loop to load
LOOPON($logs)
     $logdata = SELECT * FROM $logs.DataTable
     LOADLOG($logs,$logdata)
     CLOSE($logdata)
ENDLOOP
CLOSE($logs)
```

Include (*ScriptName (text), Prompt (bool)*)
Include (*ScriptName (text)*)

Triggers the execution of another SQL script.

Parameter:

ScriptName	Name of the script to be executed.
Prompt	If the user wants to be warned if the script specified by the *ScriptName* is not found this parameter must be set to true.

Remarks:

Allows execution of another script from the current SQL script.

Example:

```
# init tables
. . .
CLOSE($dsn)
# trigger execution of another script
INCLUDE('CreateNewProject.SQL')
```