

Inserting CELCAT® Register Marks

How to Programmatically Insert Register Marks into CELCAT Timetabler

Introduction

CELCAT *Timetabler* Attendance is used to record attendance marks on electronic registers. CELCAT provides a number of mechanisms to do this including mobile devices, a web interface, offline via data collectors, etc.

This document describes how to bypass the standard input mechanisms and insert attendance marks programmatically into the CELCAT *Timetabler*. The techniques described may be useful if you need to integrate another register marking tool or software system.

Two approaches are described:

1. *Populating the Offline Attendance staging table*. This involves writing directly to a staging table in the *Timetabler* SQL Server database. It has the advantage of being easy to implement, but the marks remain in the staging table until the CELCAT Offline Register Marking Service transforms them into register marks. This usually happens swiftly but there is always a delay which depends upon a number of factors.
2. *Using the CELCAT Timetabler COM Library*. This involves communicating with the CELCAT *Timetabler* Server (*Timetabler's* middleware component). Marks submitted in this way appear immediately in CELCAT *Timetabler* registers.

The following sections describe each method in more detail:


Populating the OLA Staging Table

The Offline Attendance (OLA) system is a means by which register marks can be collected offline (typically using a data collection device such as a barcode or MIFARE data collector) and then uploaded to the CELCAT registers. In the first instance, data is uploaded to a staging table in the CELCAT timetable database (the table is named "OLA_1"). The OLA system includes the *Register Marking Service* which periodically analyses the data in the staging table, validating the data and identifying corresponding registers. Where possible, it moves attendance marks from the staging table into the relevant CELCAT registers.

In order to use this mechanism with your independent data collection device or software system you need to understand the format of the OLA_1 staging table as described below.

Staging Table Format

The OLA_1 staging table is shown below and a description of each column follows. Note that the OLA tables (including OLA_1) are not part of the standard CELCAT timetable database and are created and managed by the Register Marking Service. So if they don't appear in your list of tables, just install and run the Register Marking Service (using the Offline Attendance Configuration program).

OLA_1			
	Column Name	Data Type	Allow Nulls
	ident	int	<input type="checkbox"/>
	session	int	<input type="checkbox"/>
	student	nvarchar(128)	<input type="checkbox"/>
	mark_stamp	datetime	<input type="checkbox"/>
	context	int	<input type="checkbox"/>
	mark_abb	nvarchar(10)	<input checked="" type="checkbox"/>
	marking_user_id	int	<input type="checkbox"/>
	status	int	<input type="checkbox"/>
	event_id	int	<input checked="" type="checkbox"/>
	week	int	<input checked="" type="checkbox"/>
	student_id	int	<input checked="" type="checkbox"/>
	mark_id	int	<input checked="" type="checkbox"/>
	dt_uploaded	datetime	<input type="checkbox"/>
	processed_count	int	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Offline Attendance Staging Table

ident is a fabricated primary key (auto-incrementing).

session is an integer value that identifies the upload session during which a mark is inserted in the table. There may be many rows for each session. The value is used by the marking service to assist in identifying groups of marks that may belong together in the same register (i.e. the service assumes that if a group of marks are recorded with similar mark_stamp values and identical session values then they probably belong to the same register).

student is the student identifier (see also 'context').

mark_stamp is the date and time at which the mark was recorded (for example, if the marks were collected using a hand-held barcode scanner then this is the actual time stamp stored against each scan).

context is an integer value used to describe the semantics of the 'student' column as follows:

- 0 = student unique name
- 1 = student name
- 2 = internal student ID
- 3 = student card number
- 4 = student original ID
- 5 = email
- 6 = profile
- 7 = lookup_id1

8 = lookup_id2

9 = lookup_id3

So, for example, if a given OLA_1 entry has a context value of 3 then the Register Marking Service knows to interpret the value in the *student* column as a student card number.

mark_abb is the mark abbreviation and corresponds to values in the CT_MARK.abbreviation column. If NULL then a 'present' mark is assumed (this is usually the one that has been specified for 'card marking', but it may also be user-specified).

marking_user_id is a reference to CT_USER.user_id and is usually the identifier of the user who marked the register (although you are free to leave it blank or specify any user identifier you wish). Note that no Foreign Key constraint is used. If you don't supply a value for the marking_user_id it defaults to the ID of the built-in administrative account.

status is an indication of the status of the row as follows:

0 = newly-inserted

The row has never been processed by the marking service. As far as the service is concerned, the row is newly inserted and therefore has priority when it begins to process rows in the OLA_1 table.

100 = examined by the service but not yet transformed to a register mark

The row has been examined by the marking service in order to establish the row's student ID (**student_id**) and mark ID (**mark_id**). Once examined, the **processed_count** column is incremented and the **dt_processed** column is set to the date/time last examined. Note that a row with 'examined' status does not necessarily mean that **mark_id** and **student_id** have been determined; if one or both column values could not be deduced they remain NULL. In this case, the marking service regularly examines the row to see if it can identify the correct **mark_id** and **student_id**. Eventually, however, the marking service will abandon a row where one or both columns cannot be deduced (see below under 'Abandoned').

200 = abandoned (after many attempts at transforming)

The row is old and has been evaluated many times by the marking service. Once abandoned it does not participate in further processing unless it is manually revived using the OLA Viewer application.

1000 = successfully transformed to a register mark

The row has been successfully transformed into a register mark. Such items are eventually removed from the **OLA_1** table when the associated register is completed.

The status column is only updated by the register marking service, and its value defaults to 0 for a newly inserted row.

The **event_id** and **week** columns are a reference to the CT_ACTIVITY table and appropriate values are set when a row is transformed to a register mark.

The **student_id** column corresponds to the internal student ID as stored in the CT_STUDENT.student_id column. It is populated as soon as the service is able to identify the student

(based on the values in student and context). It helps to identify the student as soon as possible since the value recorded in the **student** column may change between the time the data collector gathered the value and the time it is finally transformed into a register mark by the service. Note that no Foreign Key constraint is used.

The **mark_id** column corresponds to the CT_MARK.mark_id column and is populated by the service as soon as it is recognised. Note that no Foreign Key constraint is used.

dt_uploaded is the date and time when the item was uploaded to the OLA_1 table.

processed_count is used to record how many times the register marking service has analysed the item. This is useful because it allows us to identify rows that have failed many attempts at transformation.

Data to Insert

For each register mark you should insert a single row in the OLA_1 table. Mandatory and optional columns are described below along with an indication of how these might be used:

Mandatory columns are:

session – choose a session number that allows the marking service to identify related marks (e.g. if you have a data collector device you might use a combination of the device ID and the hour of the day). This is not a strict requirement, nor does it specify that all marks with a common session number belong to the same register. However, it does help the marking service to group together associated marks in an efficient way.

student and **context** – use together to identify which student was marked. For example, if you are recording IDs from MIFARE cards, and these IDs are stored in the students' card_num fields, then you should set the context value to 3 and store the ID in the student column.

mark_stamp – specify the date and time at which the register mark was taken. This is used by the marking service to identify the register against which the mark is eventually recorded. This is an important consideration in using the staging table mechanism – the marking service selects the appropriate register by finding an event that occurs at the specified mark_stamp time, and to which the specified student is assigned. During this stage, heuristics are used to analyse the data in an attempt to correctly transform the entries in OLA_1. If your system knows the register that is being marked then it may be more robust to supply the date and start time of the event rather than the date/time at which the mark was collected.

Optional columns are:

mark_abb – this defaults to a 'present' mark, but you can specify any of the mark abbreviations from the timetable's CT_MARK table.

marking_user_id – this defaults to the ID of the built-in administrative account but you can specify any *Timetabler* user ID.

student_id – the marking service normally populates this column for you based upon your entries in the **student** and **context** columns. However, if you know the CELCAT *Timetabler* internal student ID then you can insert it here.

mark_id - the marking service normally populates this column for you based upon your entry in the **mark_abb** column. However, if you know the CELCAT *Timetabler* internal mark ID then you can insert it here.

Columns that *should not* be populated are:

ident

status

event_id

week

dt_uploaded

processed_count

Typical Usage

You can insert data using SQL directly into the OLA_1 table. The following example assumes you have a networked system and a live connection to the timetable database. The student is being identified by his unique name and the current date/time is used to specify when the register mark was taken:

```
INSERT INTO OLA_1 (session, student, context, mark_stamp)
VALUES (937403, 'Jones, A P', 0, GETDATE())
```

Using the COM Library

The COM Library is a comprehensive Application Programming Interface (API) for the CELCAT *Timetabler* system. Please see the COM Library help file for full details of the functions offered.

There are several API objects of interest when interfacing to CELCAT attendance registers:

The **CTAttendanceMarks** object represents the attendance marks for a specified activity. It is used in conjunction with the **CTAttendance** and **CTAttendanceFilter** objects to provide read-only access to the CELCAT *Timetabler* Attendance marks.

The objects are generally used like this:

- Create a CTAttendance object
- Get the CTFilter object
- Specify filter parameters (staff, depts, etc)
- Open the CTAttendance object
- Iterate through the data creating a CTAttendanceMarks object where required

The above objects can only be used to *read* register data, whereas the **CTRegisterMarker** object can be used to insert marks. The general procedure to follow is:

- Identify the register to be marked
- Open the register ready for marking
- Mark the register
- Save the register

These steps are described below (the assumption is made that you understand the fundamental COM Library architecture, for which see the COM Library help file). Examples are in Visual Basic:

Identifying Registers

Timetabler (in association with its Student Attendance Tracking module) allows you to control whether or not events require registers. Where an event requires registers and runs - for example - in 10 weeks, the Attendance system generates 10 registers; 1 for each of its weeks.

If you know the event ID and week you can uniquely identify the register to mark. With this information you can call the **OpenForEventID** method specifying the event ID and week number as arguments to the function.

At other times you may wish to identify registers automatically using **OpenForRoomID** or similar functions.

Finally, you can also get a list of registers using one of the following functions. The register IDs returned from these functions can be passed to **OpenForRegID**.

GetRegistersByResource(EntityType, ID, DateTime, Result)

Gets the register IDs for a specified resource at a specified date and time.

Parameters:

EntityType (Integer). Entity type, e.g. CTLIB_ROOM, CTLIB_STAFF, etc

ID (Integer). ID of the resource

DateTime (DATE). Date and time for the register (see notes)

Result (String Ref). Return value - a comma-separated list of register IDs

Example:

```
Dim RegM As CTCOMLib.ICTRegisterMarker = Session.RegisterMarker
Dim ids As String =
    RegM.GetRegistersByResource(CTLIB_ROOM, roomID,
    DateValue("17 June 2008") + TimeValue("4:05:00 PM"))
```

Notes:

You can extract individual register IDs from the comma-separated list returned from this function. The register IDs can be used in calls to **OpenForRegID**.

The DateTime value should be set carefully. The function searches for registers that are current at the specified date and time (i.e. the activity is actually taking place at the specified date and time; DateTime falls between the activity start and the activity end). For most resources, this should limit the result to a single register at most.

GetRegistersByRoom(ID, DateTime, Result)

Gets the register IDs for a specified room at a specified date and time.

Parameters:

ID (Integer). ID of the room

DateTime (DATE). Date and time for the register

Result (String Ref). Return value - a comma-separated list of register IDs

Notes:

You can extract individual register IDs from the comma-separated list returned from this function. The register IDs can be used in calls to **OpenForRegID**.

The DateTime value should be set carefully. The function searches for registers that are current at the specified date and time (i.e. the activity is actually taking place at the specified date and time; DateTime falls between the activity start and the activity end).

GetRegistersByStaff(ID, DateTime, Result)

Gets the register IDs for a specified member of staff at a specified date and time.

Parameters:

ID (Integer). ID of the member of staff

DateTime (DATE). Date and time for the register

Result (String Ref). Return value - a comma-separated list of register IDs

Notes:

You can extract individual register IDs from the comma-separated list returned from this function. The register IDs can be used in calls to **OpenForRegID**.

The DateTime value should be set carefully. The function searches for registers that are current at the specified date and time (i.e. the activity is actually taking place at the specified date and time; DateTime falls between the activity start and the activity end).

GetRegistersByStudent(ID, DateTime, Result)

Gets the register IDs for a specified student at a specified date and time.

Parameters:

ID (Integer). ID of the student

DateTime (DATE). Date and time for the register

Result (String Ref). Return value - a comma-separated list of register IDs

Notes:

You can extract individual register IDs from the comma-separated list returned from this function. The register IDs can be used in calls to **OpenForRegID**.

The DateTime value should be set carefully. The function searches for registers that are current at the specified date and time (i.e. the activity is actually taking place at the specified date and time; DateTime falls between the activity start and the activity end).

Opening Registers for Marking

There are several methods for opening a register ready for marking:

OpenForEventID(EventID, Week)

Opens the specified register.

Parameters:

EventID (Integer). ID of the event associated with the register

Week (Integer). Week of the event

Notes:

Week is 1-based (i.e. the first week in the timetable is considered week 1) and should not be confused with any week numbering scheme you may have established.

OpenForRegID(RegID)

Opens the specified register.

Parameters:

RegID (Long Integer). ID of the register

Notes:

You can use the functions described in Identifying Registers to search for relevant register IDs.

Although you probably won't need to generate register IDs, they are fabricated using an event ID and week number as follows:

```
Dim RegID As Long = (eventID * 100) + wk
```

OpenForResourceID(Type, ID, DateTime)

Opens the register associated with a specified resource at a specified date and time.

Parameters:

Type (Integer). Entity Type of the resource, e.g. CTLIB_ROOM, etc

ID (Integer). Resource ID

DateTime (DATE). Date and time of the register

Notes:

The assumption here is that the resource can only be involved in one activity at a time (although this is not enforced in *Timetabler*). Thus, you only need to specify the resource and a date/time in order to open the relevant register.

The DateTime value should be set carefully. The function searches for registers that are current at the specified date and time (i.e. the activity is actually taking place at the specified date and time; DateTime falls between the activity start and the activity end).

You may want to catch the following error conditions:

CTLIB_ERR_REG_NOT_EXISTS - no register exists for this resource at the date and time specified.

CTLIB_ERR_MULTIPLE_REGISTERS - multiple registers exist for this resource at the date and time specified.

In either case the register is not opened.

OpenForRoomID(ID, DateTime)

Opens the register associated with a specified room at a specified date and time.

Parameters:

ID (Integer). Room ID

DateTime (DATE). Date and time of the register

Notes:

The assumption here is that the room can only be used for one activity at a time (although this is not enforced in *Timetabler*). Thus, you only need to specify the room and a date/time in order to open the relevant register.

The DateTime value should be set carefully. The function searches for registers that are current at the specified date and time (i.e. the activity is actually taking place at the specified date and time; DateTime falls between the activity start and the activity end).

You may want to catch the following error conditions:

CTLIB_ERR_REG_NOT_EXISTS - no register exists for this room at the date and time specified.

CTLIB_ERR_MULTIPLE_REGISTERS - multiple registers exist for this room at the date and time specified.

In either case the register is not opened.

OpenForStaffID(ID, DateTime)

Opens the register associated with a specified member of staff at a specified date and time.

Parameters:

ID (Integer). Staff ID

DateTime (DATE). Date and time of the register

Notes:

The assumption here is that the member of staff can only be used for one activity at a time (although this is not enforced in Timetabler). Thus, you only need to specify the staff ID and a date/time in order to open the relevant register.

The DateTime value should be set carefully. The function searches for registers that are current at the specified date and time (i.e. the activity is actually taking place at the specified date and time; DateTime falls between the activity start and the activity end).

You may want to catch the following error conditions:

CTLIB_ERR_REG_NOT_EXISTS - no register exists for this member of staff at the date and time specified.

CTLIB_ERR_MULTIPLE_REGISTERS - multiple registers exist for this member of staff at the date and time specified.

In either case the register is not opened.

OpenForStudentID(ID, DateTime)

Opens the register associated with a specified student at a specified date and time.

Parameters:

ID (Integer). Student ID

DateTime (DATE). Date and time of the register

Notes:

The assumption here is that the student can only be used for one activity at a time (although this is not enforced in Timetabler). Thus, you only need to specify the student and a date/time in order to open the relevant register.

The DateTime value should be set carefully. The function searches for registers that are current at the specified date and time (i.e. the activity is actually taking place at the specified date and time; DateTime falls between the activity start and the activity end).

You may want to catch the following error conditions:

CTLIB_ERR_REG_NOT_EXISTS - no register exists for this student at the date and time specified.

CTLIB_ERR_MULTIPLE_REGISTERS - multiple registers exist for this student at the date and time specified.

In either case the register is not opened.

Navigating

GetCount(Value) and Count

Gets the number of students in the register.

First()

Goes to the first student record.

Last()

Goes to the last student record.

Next()

Goes to the next student record.

Prior()

Goes to the previous student record.

GetAtEnd(Value) and AtEnd

Determines if the current position is at the end of the set.

Parameters:

Value (Boolean Ref). Return value - True if at end

Example:

```
Dim RegM As CTCOMLib.ICTRegisterMarker = Session.RegisterMarker
RegM.OpenForRoomID(roomID, DateValue("17 June 2008") + TimeValue("10:05:00 AM"))
While Not RegM.AtEnd
    MsgBox(RegM.Mark)
End While
```

Notes:

This method can be used to assist in looping through records or to determine if the set is empty when first opened.

Getting Values

GetEventID(Value) or EventID

Gets the event ID of the current record.

Parameters:

Value (Integer Ref). Return value - event ID

Notes:

Since the **CTRegisterMarker** object opens a single register at a time, the event ID will be the same for all records in the set.

GetWeek(Value) or Week

Gets the week number of the current record.

Parameters:

Value (Integer Ref). Return value - week

Notes:

The week number is 1-based (i.e. the first week is 1) Do not confuse these week numbers with any numbering scheme you may have configured in *Timetabler*.

GetStudentID(Value) or StudentID

Gets the student ID of the current record.

Parameters:

Value (Integer Ref). Return value - student ID

GetStamp(Value) or Stamp

Gets the date/time stamp of the current record (the date and time when the mark was recorded).

Parameters:

Value (DATE Ref). Return value - date/time stamp

Notes:

If there is no mark for the current record then the Stamp value is not meaningful.

GetMark(Value) or Mark

Gets the name of the mark associated with the current record.

Parameters:

Value (String Ref). Return value - name of mark

Notes:

The mark name is empty if there is none.

GetMarkID(Value) or MarkID

Gets the mark ID of the current record.

Parameters:

Value (Integer Ref). Return value - mark ID

Notes:

The mark ID is zero if there is currently no mark. See also `CTMarkTypes`

GetMinsLate(Value) or MinsLate

Gets the number of minutes late for the current mark.

Parameters:

Value (Integer Ref). Return value - minutes late

Notes:

The value is only meaningful if the mark has a 'Late' definition.

GetComments(Value) or Comments

Gets any comments associated with the current record.

Parameters:

Value (String Ref). Return value - comments

GetMarkShortcut(Value) or MarkShortcut

Gets the current mark as a shortcut value.

Parameters:

Value (String Ref). Return value - shortcut

Notes:

See also `CTMarkTypes`

Marking**MarkStudent(StudentID, MarkID, MinsLate, Comments)**

Marks the specified student using the specified mark.

Parameters:

StudentID (Integer). Student ID

MarkID (Integer). Mark ID

MinsLate (Integer). Minutes late

Comments (String). Textual comments

Notes:

MinsLate will be ignored if the mark is not a 'Late' type.

The function raises an error if it can't find the specified student on the register.

See also **CTMarkTypes**

SetMark(Value) or Mark

Sets the current Mark.

Parameters:

Value (String). Name of the mark

Notes:

See also **SetMarkID** to mark the current record using a mark ID rather than a mark name.

SetMarkID(Value) or MarkID

Sets the current Mark

Parameters:

Value (Integer). Mark ID

SetMinsLate(Value) or MinsLate

Sets the number of minutes late.

Parameters:

Value (Integer). Minutes late

Notes:

The value is ignored if the mark is not a 'Late' type. This means that if you wish to specify a late mark, ensure that you change the **MarkID** first before changing the **MinsLate** value.

SetComments(Value) or Comments

Sets the comments associated with the current record.

Parameters:

Value (String). Textual comments

SetMarkShortcut(Value) or MarkShortcut

Sets the current Mark.

Parameters:

Value (String). Shortcut value of the mark

Notes:

The shortcut value is the shortcut key that is generally used for marking registers in the Windows or Web client.

Save()

Saves the changes to the current register.

Querying

IsStudentOnReg(StudentID, Result)

Determines if a student is on the register

Parameters:

StudentID (Integer). Student ID

Result (Boolean Ref). Return value - true if student is on register

GetRegDateTime(Start, End) or RegDateTimeStart/RegDateTimeEnd

Gets the date and time of the register (i.e. the date and time at which the activity took place).

Parameters:

Start (DATE Ref). Return value - start of activity

End (DATE Ref). Return value - end of activity

GetStudentIDList(Value)

Gets a CSV list of student IDs for students on the register.

Parameters:

Value (String Ref). Return value - CSV list of student IDs

GetStudentMark(StudentID, MarkID, MinsLate, Comments)

Gets details of a student's mark on the register.

Parameters:

StudentID (Integer). Student ID

MarkID (Integer Ref). Return value - student's mark ID

MinsLate (Integer Ref). Return value - minutes late

Comments (String Ref). Return value - comments

Notes:

This function raises an error if the student doesn't exist on the register.

MinsLate is only applicable if the mark is a 'Late' type.

GetActivityID(Value) or ActivityID

Gets the activity ID for the register.

Parameters:

Value (Integer Ref). Return value - activity ID

FindByStudentID(ID, Found)

Finds a record in the register by student ID.

Parameters:

ID (Integer). Student ID

Found (Boolean Ref). Return value - true if found

GetMarkSource(Value) or MarkSource

Gets the source of the mark.

Parameters:

Value (Integer Ref). Return value - source (see in Notes)

Notes:

Possible values of source are:

CTLIB_MST_REGISTER - a normal register mark (entered by a user when marking a register)

CTLIB_MST_EXCEPTION - a mark that results from a student exception list (e.g. when a student has been withdrawn from a course)

CTLIB_MST_EXTENDED_ABSENCE - a mark that results from an extended absence record (e.g. student holiday, illness, etc)

CTLIB_MST_UNKNOWN