

Database API Enhancements

This document discusses the changes to the Database API included with Commence RM.

1 ICommenceConversation

1.1 Overview

This object is accessed from the Commence.DB object using the GetConversation() method, which returns an ICommenceConversation object. This object represents a pseudo "DDE" conversation with Commence, and can be used to send any of the DDE commands supported by Commence. [See the DDE.HLP for command syntax.]

1.2 Reference

The following lists the new methods added to the Database API to support the Conversation object. [These methods are also documented in the DBAPI.HLP file.]

1.2.1 ICommenceDB::GetConversation Method

ICommenceConversation GetConversation(String *pszApplicationName*, String *pszTopic*)

Create a conversation object.

Returns a pointer to conversation object on success, NULL on error.

pszApplicationName

DDE Application name.

The only allowed value is "Commence".

pszTopic

DDE Topic name, must be a valid Commence topic name.

The DDE.HLP file contains a reference to the allowed DDE topic names.

Examples include "GetData", "ViewData", etc.

1.2.2 ICommenceConversation

Represents a DDE conversation.

Boolean Execute(String *pDDECommand*)

Executes the DDE Command.

String Request(String *pDDECommand*)

Processes the DDE Request.

1.2.3 ICommenceConversation::Execute Method

Boolean Execute(String *pDDECommand*)

Executes the DDE Command.

Returns TRUE on success, FALSE on error.

pDDECommand

Text with the DDE command. Syntax is identical to the commands used by the DDE API.

TBD.

Return Value
Parameters

Description
Methods

Return Value
Parameters

Comments

1.2.4 ICommenceConversation::Request Method

String Request(String pDDECommand)

Processes the DDE Request.

Return Value
Parameters

Returns String value on success.

pDDECommand

Text with the DDE Request. Syntax is identical to the commands used by the DDE API.

Comments

TBD.

1.3 Sample Code

The following sample code shows how to use the Conversation object.

1.3.1 Getting Commence Preferences

The following code shows how the Conversation object can be used to query the Commence preference for the External Files folder. The DDE command: “[GetPreference(ExternalDir, |)]” will be sent via the Request() method.

```
Set obj = CreateObject("Commence.DB")
set conv = obj.GetConversation("Commence", "GetData")
str = conv.Request("[GetPreference(ExternalDir, |)]")
MsgBox "External Files Folder: " & str
` str contains the path to the External Files folder
```

1.3.2 Open a Commence view

` Show view on Commence desktop

```
Set obj = CreateObject("Commence.DB")
set conv = obj.GetConversation("Commence", "GetData")

dim flag
flag = conv.Execute("[ShowView("Contacts", 1)]")
if CBool(flag) then
    MsgBox "Started the Contacts View"
else
    MsgBox "Could not start the Contacts View"
end if

set conv = nothing
set obj = nothing
```

1.3.3 List fields in a Commence category

```
` get list of tables
Set obj = CreateObject("Commence.DB")
set conv = obj.GetConversation("Commence", "GetData")
str = conv.Request("[GetFieldNames(Person/Company, |)]")
set conv = nothing
set obj = nothing
MsgBox "Person/Company Field Names: " & vbcrLf & vbcrLf & str
```

2 New Constants/Flags

The following flags have been added for use with the Database API.

2.1 CMC_FLAG_USETHIDS

The CMC_FLAG_USETHIDS flag is used with the GetCursor() method and causes the database API to return workgroup THIDs for RowIDs and for use when working with connection fields. This flag will not work with all databases: only workgroup and sync-link databases have THIDs; stand-alone databases do not have THIDs, and this flag cannot be supported.

If the GetCursor() method determines that the CMC_FLAG_USETHIDS flag cannot be supported, it will throw an exception to inform the caller of the error. This exception must be trapped, and if appropriate, the GetCursor() method can be called again without this flag.

2.1.1 CMC_FLAG_USETHIDS = &H0040

Used with the CURSOR object. Indicates that the Rowset objects should return THID ids, and accept THID ids. See GetRowID, GetRowValue, ModifyRow, Get*RowSetByID() methods.

2.1.2 Working with the CMC_FLAG_USETHIDS flag

When the CMC_FLAG_USETHIDS flag is used and accepted on the GetCursor() method, the following are enabled:

- the GetRowID() method will return ids (datatype: string) based on THIDs, instead of a pseudo-rowid that is randomly generated
- the ModifyRow() method will accept the RowID string when used with a connection field in order to connect to the proper database record
- the GetRowValue() method when used with a connection field will return the row-ids for the connected items, instead of the Name field of the connected items, in a comma-delimited list.
- The Get*RowSetByID() methods require that the RowID parameter be the THID-based id. For example, GetQueryRowSetByID() will accept the THID-based RowID string to access the proper database record.

2.1.3 Example

Assume that this cursor has only a connection field.

‘ the following should return a comma-delimited list of THIDs, eg. “0:01:0005C, 0:01:0005D, 0:01:0005E” instead of returning the Name field values in comma-delimited list, eg. “name1, name2, name3”

```
str1 = qrs1.GetRowValue(0,0,0)
```

‘ this method should accept the string as a THID eg, “0:01:0005C” instead of using the Name field value, eg. “name1”

```
str1 = qrs1.ModifyRow(0, 0, str, flags)
```

2.1.4 Sample code with CMC_FLAG_USETHIDS flag

```
dim usethids
usethids = 256 `CMC_FLAG_USETHIDS
dim ignoresync
ignoresync = 512 `CMC_FLAG_IGNORESYNCCONDITION

set contacts = cmc.GetCursor(7, "",usethids+ignoresync)
if Err.Number <> 0 then
    MsgBox "Cannot use THIDS"
    Exit
End if

` search by email field
contacts.setfilter("[ViewFilter(1, F,, E-mail, Equal To, <email address>,)]")

if contacts.RowCount <= 0 then
    MsgBox "Contact not found"
End if
`TODO: to handle multiple contacts, put this into a for loop

` get 1 row from the cursor into a queryrowset object
qrs = contacts.GetQueryRowset(1, 0)

` row id is returned as THID-based ids in a string data format
rownumber = 0
strRowid = qrs.GetRowID(rownumber, 0)

` create new Email log record
set emaillog = cmc.GetCursor(9, "",usethids)
set addrowset = emaillog.GetAddRowSet(1, CMC_FLAG_SHARED)

` update the `for contact` column with the contact's rowed
colnumber = addrowset.GetColumnIndex("For Contact", 0)
addrowset.ModifyRow(rownumber, colnumber, strRowid, 0)

` if there are multiple contacts, call ModifyRow for each one
addrowset.ModifyRow(rownumber, colnumber, strRowid1, 0)
addrowset.ModifyRow(rownumber, colnumber, strRowid2, 0)

`TODO: update other columns in this new record
```

2.1.5 Sample code without the CMC_FLAG_USETHIDS flag

When the CMC_FLAG_USETHIDS flag is not used or not accepted on the GetCursor() method, the existing RowIDs from Commence 2000 are used:

- the GetRowID() method will return ids (datatype: string) based on a pseudo-rowid that is randomly generated and valid only for this session
- the ModifyRow() method will accept only the Name field value for records to be connected, which may not be unique
- the GetRowValue() method when used with a connection field will return the Name field value for all connected items, delimited by commas (,)
- The Get*RowSetByID() methods will use the pseudo- RowID parameter returned from GetRowID().

2.1.6 Pseudo-code without USETHIDS flag

```
dim ignoresync
ignoresync = 512 `CMC_FLAG_IGNORESYNCCONDITION
```

```

set contacts = cmc.GetCursor(7, "", ignoresync)

` search by email field
contacts.setfilter("[ViewFilter(1, F,, E-mail, Equal To, <email address>,)]")

if contacts.RowCount <= 0 then
    MsgBox "Contact not found"
End if
`TODO: to handle multiple contacts, put this into a for loop

` get 1 row from the cursor into a queryrowset object
qrs = contacts.GetQueryRowset(1, 0)

` get the "name" field for this row
colnumber = qrs.GetColumnIndex("File As*", 0)
rownumber = 0
strName = qrs.GetRowValue(rownumber, colnumber, 0)

` create new Email log record - don't set the USETHIDS flag
set emaillog = cmc.GetCursor(9, "", 0)
set addrowset = emaillog.GetAddRowSet(1, CMC_FLAG_SHARED)

` update the 'for contact' column with the contact's name
colnumber = addrowset.GetColumnIndex("For Contact", 0)
addrowset.ModifyRow(rownumber, colnumber, strName, 0)

` if there are multiple contacts, call ModifyRow for each one
addrowset.ModifyRow(rownumber, colnumber, strName1, 0)
addrowset.ModifyRow(rownumber, colnumber, strName 2, 0)

`TODO: update other columns in this new record

```

2.2 CMC_FLAG_IGNORESYNCCONDITION

The Commence Other Apps preference allows the user to map from a Commence table to a “pseudo-table” which can be accessed from the Database API. For example, the Outlook Contacts mapping can be accessed by using the special cursor type:

```
#define CMC_CURSOR_OUTLOOKAB 7 // MS Outlook Address book
```

The preference also includes a “Sync Condition” mapping that is used to include only a subset of the items in the database. At times, it is necessary to ignore this sync condition and consider all items in the mapped Commence category. In order to support this, the CMC_FLAG_IGNORESYNCCONDITION has been added. This flag can be used with the GetCursor() method and lets the API know that all records in the Commence category should be considered.

2.2.1 CMC_FLAG_IGNORESYNCCONDITION= &H0200

Used with the CURSOR object. Indicates that the Rowset objects should return THID ids, and accept THID ids. See GetRowID, GetRowValue, ModifyRow, Get*RowSetByID() methods.

2.2.2 Sample

```

dim ignoresync
ignoresync = 512 `CMC_FLAG_IGNORESYNCCONDITION

set contacts = cmc.GetCursor(7, "", ignoresync)
` use the contacts cursor to filter, etc.

```

3 Outlook E-mail Log Pseudo-table

The following describes the fields in the pseudo-table for Email logging, including the expected use and expected data type.

3.1 Pseudo-fields

The following are the field names in the Email Logging pseudo-table, their purpose, and data types.

3.1.1 Attachment

Mapped to the OL “Attachment” field

This field indicates whether the message has any attachments. It can be mapped to a commence checkbox field which accepts “1” for true and “0” for false.

3.1.2 Bcc

This field receives the Outlook “Bcc” field.

This is a text field that receives the list of names in a comma-delimited list.

3.1.3 Body

This field receives the Outlook “Message” field.

This is a large text field that receives the ASCII text of the email message body.

3.1.4 Categories

Mapped to the OL “Categories” field

This is a text field that receives the comma-delimited list of categories for the message.

3.1.5 Cc

This field receives the Outlook “Cc” field.

This is a text field that receives the list of names in a comma-delimited list.

3.1.6 Contact E-mail

This is a text field that is set with the e-mail address of the contact. For incoming messages, this is the sender email address. For outgoing messages, this is the recipient email address.

Note: for outgoing mail, this could be multiple contacts. The email address for each contact should be added in a comma-delimited list.

3.1.7 Contact Name

This text field receives the Outlook “From” or Outlook “To” field, depending on whether this is an incoming or outgoing message.

This may be the email address or display name or combination of both.

Note: for outgoing mail, this could be multiple contacts. The email address for each contact should be added in a comma-delimited list.

3.1.8 Folder Name

Mapped to the OL “In Folder” field

This is a text field that receives the name of the folder where the message is stored.

3.1.9 Follow Up Note

Mapped to the OL “Follow Up Flag” field

This is a text field that receives the string describing the follow up action.

3.1.10 For Contact

This field is used to relate the message with the appropriate contacts and is mapped to a Connection field in Commence.

- For incoming mail, this will be set with the list of sender(s)
- For outgoing mail, this will be set with the list of recipient(s).

3.1.11 Importance

This field receives the Outlook “Importance” field.

This is a text field.

3.1.12 Message Date/Time

This set of fields is used for both incoming and outgoing messages, but is set from different Outlook message fields based on whether the message is incoming or outgoing.

- For incoming messages, the message date/time are set to the Outlook received date/time.
- For outgoing messages, the message date/time are set to the Outlook Sent date/time.

3.1.12.1 Message Date

This field receives the date portion of the Outlook “Received” or “Sent” field, for incoming and outgoing messages, respectively.

3.1.12.2 Message Time

This field receives the time portion of the Outlook “Received” or “Sent” field, for incoming and outgoing messages, respectively.

3.1.13 Message File

This is a text or data-file field receives the path and filename of the .MSG file that is created with the entire Outlook email message, including rich text formatting and attachments.

3.1.14 Message Size

Mapped to the OL “Size” field

This is a text or number field that receives the size of the message as a number, for example, 8192.

3.1.15 Message Type: “E-mail Received”, “E-mail Sent”

This field is used to set the log entry type, that it is an email and that it was sent or received.

This is a text or selection/combo-box field that receives the following:

- the label “E-mail Received” for received messages
- the label “E-mail Sent” for sent messages

3.1.16 Modification Date/Time

Outlook supports date/time values in a single field whereas Commence separates dates and times into individual field.

3.1.16.1 Modification Date

This field receives the date portion of the Outlook “Modified” field.

3.1.16.2 Modification Time

This field receives the time date portion of the Outlook “Modified” field.

3.1.17 Sensitivity

This field receives the Outlook “Sensitivity” field.

This is a text field that receives a string with the sensitivity defined by Outlook.

3.1.18 Subject

This field receives the Outlook “Subject” field.

This is a text field that receives the subject text for the message.