BMG LABTECH

ActiveX and DDE Interface Manual

for software versions 1.01 … 5.10

TABLE OF CONTENTS

1 Introduction

1.1 Overview

The Omega software has a built-in ***ActiveX interface*** and a ***DDE interface***. Both interfaces provide the same set of high level commands, which enable your software to remote-control the FLUOstar Omega, LUMIstar Omega, POLARstar Omega, SPECTROstar Omega or NEPHELOstar Plus reader very easily.

The Omega software (Omega.exe) acts as a *ActiveX or DDE server* while your own software acts as a *client*. The Omega software must be active the entire time!

1.2 ActiveX Interface Description

The BMG LABTECH ActiveX Automatization Interface will be provided as ocx-file (In Proc server). The ocx-component will be installed and registered together with the reader control software.

The ActiveX automatization interface contains the following methods:

OpenConnection (ServerName: PChar; out Result: OleVariant)

This method will start the ActiveX server. If the corresponding reader control program is not yet running it will be started in minimized (iconized) state and the reader connected will be initialized. (If the reader control program has already be started the reader will not be initialized again.)

The method expect as parameter (Pascal type: PChar, IDL type: LPSTR) the name of the BMG LABTECH reader control program to be used as ActiveX server, e.g. ‘Omega’. If multiple installations of a program exists, you can include an installation number, e.g. ‘Omega3’.

The function will return 0 if it has been executed successfully (Pascal type OleVariant, IDL type VARIANT *). If there is already an open connection using the same server name the return value will be -1. If a different server is active -3 will be returned. To change the ActiveX server please close an existing connection (using *CloseConnection*, see below) before using this function to establish a new connection. If the specified ActiveX server is not installed (not registered) the return value will be -2.

GetVersion (out Value: OleVariant)

This method will retrieve the version of the BMG LABTECH Remote Control ActiveX component.

GetInfo (ItemName: PChar; out Value: OleVariant)

This method will retrieve the value of the specified item. As ItemName (Pascal type: PChar, IDL type: LPSTR) you can use all items described in chapter 2. The return value of Pascal type OleVariant (IDL type VARIANT *) will contain one string value.

If the function could not been executed because the connection has not yet been opened (using the *OpenConnection* function), the return value will be “Error: -1”.

Note: If you specify a non-existing item name, the function will return an empty string.

Execute (var CmdAndParameter: OleVariant; out Result: OleVariant)

Use this method to send a command to the reader. The method will return immediately after sending the command (will not wait until executing the command has been finished). As parameter the command name including (optional) command parameters is expected (see chapter 3). The parameter type is OleVariant (IDL: VARIANT *), containing an array of values.

Example:

```
CmdAndParameter[0]:=’PlateOut’;
CmdAndParameter[1]:=’User’;
CmdAndParameter[2]:=-20;
CmdAndParameter[3]:=4280;
```

The function will return 0 if it has been executed successfully (Pascal type OleVariant, IDL type VARIANT *). Otherwise the following return codes will be send:

-1: Connection to the reader control program has not been established (no server name provided) → use OpenConnection.

-2: Connection to the reader control program has not been established (OpenConnection was not successful).

-3: Command could not be send as the connection to the reader control program was lost, reopening the connection failed.

-4: Command could not be send due to any other reason.

Note: If you send a non-existing command or a command with invalid parameters, the return value will be 0 (as the command has been sent successfully), but the information item 'Status' will change to "Error". In these cases you can get an error message (e.g. "ActiveX: Unknown command") via the item 'Error' using the *GetInfo* function.

ExecuteAndWait (var CmdAndParameter: OleVariant; out Result: OleVariant)

This method will send a command to the reader and wait until the execution of the command has been finished.

The function will return 0 if it has been executed successfully (Pascal type OleVariant, IDL type VARIANT *). Otherwise the following return codes will be send:

-1: Connection to the reader control program has not been established (no server name provided) → use OpenConnection.

-2: Connection to the reader control program has not been established (OpenConnection was not successful).

-3: Command could not be send as the connection to the reader control program was lost, reopening the connection failed.

-4: Command could not be send due to any other reason.

-10: There has been a timeout during waiting for the instrument status to change to 'Ready'.

-11: There has been a timeout during waiting for the instrument status to change to 'Busy'.

-20: There has been an error caused by sending a non-existing command or a command with invalid parameters. In these cases you can get an error message (e.g. "ActiveX: Unknown command") via the item 'Error' using the *GetInfo* function.

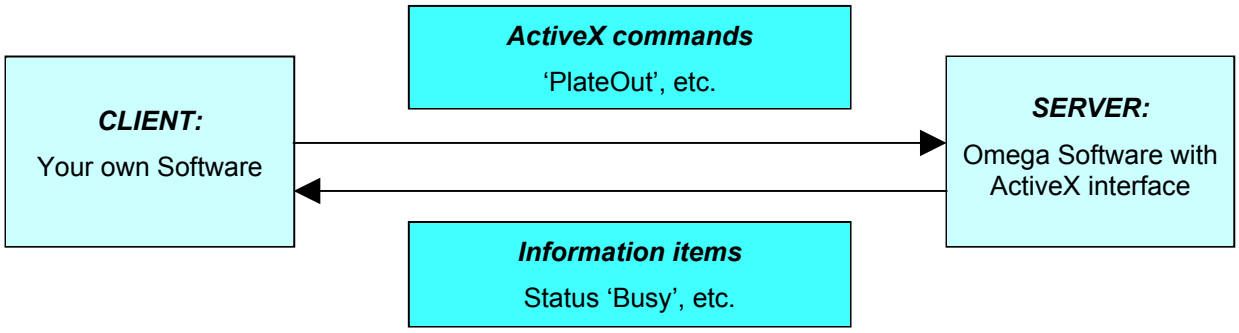
Note: The ExecuteAndWait method is only intended for those commands, which really cause a reader action, as it waits for instrument status changes. Therefore, this method should not be used for the following commands: ClearDilutionFactors, ClearSampleIDs, SetGain, SetSampleIDs, Terminate and User.

CloseConnection

This function will initialize the reader and afterwards will close the reader control program and terminate the ActiveX server.

Note: CloseConnection will automatically also send the Terminate command (see chapter 3.18), therefore, it is not necessary to send a Terminate command when ending an ActiveX session. On the other hand: by only sending the Terminate command and not using the CloseConnection procedure the configuration file settings (see chapter 1.3) will not be reset. Therefore, this is not recommended.

Example for ActiveX communication



Your own client software sends a plate out request (ActiveX command: 'PlateOut Normal') to the Omega server software.

- The Omega software receives the plate out request from the client and processes this command.
- The Omega server software sends the status information back to the client (Information items: Status = Busy, PlateOut = 0 ... Status = Ready, PlateOut = 1)

1.3 DDE Interface Description

To use the Omega software as a DDE server, you should change the value of the '**AsDDEServer**' parameter in the Omega configuration file Omega.ini (section [ControlApp]) to 'true'. This file is found in the Omega main directory (usually C:\Program Files\BMG\Omega). The default value of this parameter is 'false'. Changing this parameter to true will cause the Omega software to start iconized in 'DDE-Server mode'. In this mode, only critical error messages are shown; that means errors which no longer allow normal measurement (like hardware errors). By clicking on the program icon / name in the windows task bar you can switch from iconized mode to normal mode.

Tip: It is a good idea to read the value of the AsDDEServer parameter during the startup of your client program, save this value, set the value to 'true' and write the saved original value back when you terminate your program. To find the location of the Omega configuration file you should read the key 'HKEY_LOCAL_MACHINE\SOFTWARE\BMG Labtechnologies\Omega\1\ControlApp\MainDir' from the windows registry.

In addition, you can disable the plate in and out buttons of the software and the reader by setting the configuration file parameter '**DisablePlateCmds**' to 'true' (section [ControlApp]). Please reset this parameter to 'false' when closing the connection.

Note: If you use the ActiveX interface (and not the DDE interface), the necessary settings of the configuration file parameters will be done automatically as part of the OpenConnection and the CloseConnection procedure.

The DDE-interface consists of 2 main parts:

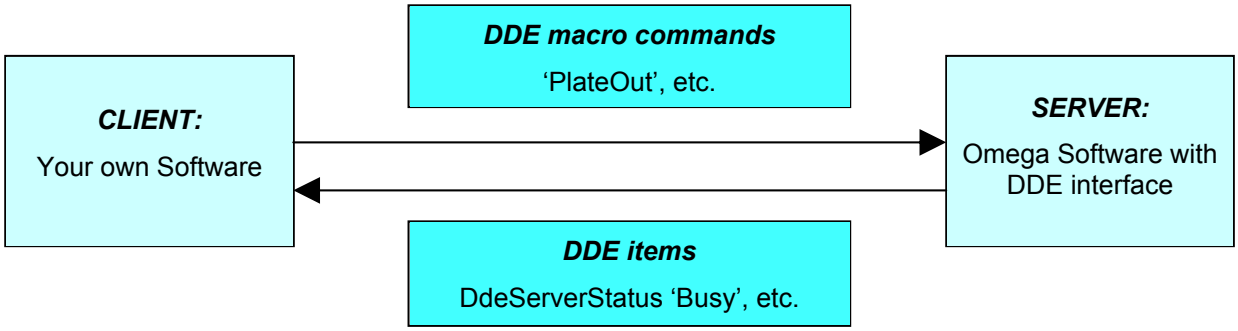
- Reception of Control commands (**DDE macro commands**)
- Transmission of status information (**DDE items**)

Each DDE information can be obtained from a Windows client program defining DDE service, DDE topic and DDE item. How to define it depends on your particular development system. An example, written in Delphi, is shown at the end of this document.

Your DDE service (**DDE application name**) is *'**Omega**'*.

The **DDE topic** will always be *'**DDEServerConv1**'*.

Example for DDE communication



Your own client software sends a plate out request (DDE macro command: 'PlateOut Normal') to the Omega server software.

- The Omega software receives the plate out request from the client and processes this command.
- The Omega server software sends the status information back to the client (DDE items: DdeServerStatus = Busy, DdeServerPlateOut = 0 ... DdeServerStatus = Ready, DdeServerPlateOut = 1)

1.4 Executing Commands

To execute a command (e.g. the Run command, see chapter 3.10), you should implement the following steps:

- wait for the device status (item ‘Status’ / ‘DdeServerStatus’, see chapter 2) to become 'Ready' (which means the reader is no longer busy executing the last command)
- send the command (see chapter 3)
- wait for the device status to become ‘Busy’ or 'Running' (or 'Error', in this case show an error message)
- wait for the device status to become 'Ready' (which means the reader has finished executing the command)
- continue with next command

Do not use fixed time values between sending commands, instead wait for the reader status to become ‘Ready’ before sending the next command. For example processing a plate command can last shorter or longer depending on where the plate carrier was before the command and depending on whether the usage of the microplate sensor has been switched on or not.

After sending a run command the time until the reader status changes to ‘Running’ depend on the used test protocol. When processing a complex 1536 well protocol or a 384 well protocol with individual injection volumes it will last longer for the status to change to ‘Running’ (or ‘Error’) than when processing a simple endpoint protocol.

Notes: Before sending commands you need to open the connection (using the OpenConnection function of the ActiveX interface or by using the `DDEClient.SetLink` function and sending a ‘Dummy’ DDE command, see chapters 7 and 8). After opening the connection the reader control software will initialize the reader (which lasts a few seconds). After this it will send a second command to transfer the reader EEPROM content to the computer. Please wait until those two commands are finished (wait for the device status to become ‘Ready’ for more than a fraction of a second) before sending commands by yourself.

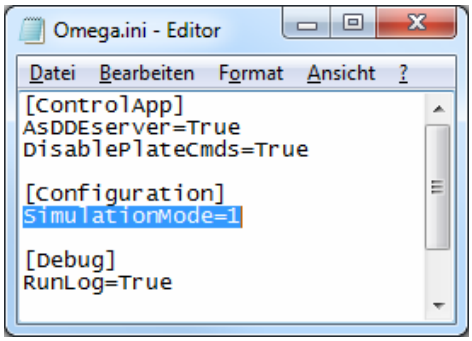
You should also use a time-out. There will be, for example, no ‘Busy’ (or only for a very short time) after sending a plate out command if the reader plate carrier is already at the requested position.

When using the ActiveX interface you can use the ExecuteAndWait function instead of the Execute function. ExecuteAndWait contains the necessary steps to wait until executing the command has been finished.

1.5 Simulation Mode

The Omega software contains a feature call ‘Simulation Mode’, which allows you to perform some tests of your ActiveX or DDE client software without using a FLUOstar / LUMIstar / POLARstar / SPECTROstar Omega or NEPHELOstar Plus reader. To switch on the simulation mode, add a line ‘SimulationMode=1’ to the section [Configuration] of the Omega configuration file ‘Omega.ini’. You will find this file in the Omega main directory (usually c:\Program Files\BMG\Omega). You can use the key combination [Shift]+[Ctrl]+[I] inside the main screen of the control software to open the configuration file.

If the section [Configuration] does not yet exists simply add it at the end of the file.



When the simulation mode is active, the Omega software simulates the reaction of an ActiveX or DDE command without communicating to a real reader, e.g. after sending a ‘PlateIn’ command the ‘Status’ item will change for a short time to ‘Busy’ and then back to ‘Ready’.

2 Status Information

The following table lists all available ActiveX or DDE items (prepared by the server program) and their possible values.

All strings are considered to be **case sensitive.**

When using the DDE interface you need to add a **prefix** '**DdeServer**' to all items besides StackerStatus and StackerKindOfResponse, e.g. use 'DdeServerStatus' instead of 'Status'. Instead of StackerStatus use DdeStackerStatus and instead of StackerKindOfResponse use DdeStackerKindOfResponse

Item	Information provided
Status	**actual device status** - 'Ready': instrument is in standby state and all measurement data has been transferred - 'Busy': instrument is busy (but no test run active) - 'Running': test run (measurement) is currently performed - 'Pausing': instrument is in pause state - 'Hardware Error': a hardware error occurred - 'Error': error occurred * * The error state will be reset using the ResetError command or using any other command besides Dummy and MotorDis / MotorEn.
DeviceConnected	**device connected *** - '0' : no reader connected - '1' : a reader has been connected * available beginning with software version 3.00
DeviceInitialized	**device initialized *** - '0' : the reader has not been initialized (no init command performed) - '1' : the reader has been initialized * available beginning with software version 3.00
DeviceBusy	**device busy** - '0' : instrument in standby (ready for commands) - '1' : instrument busy (command is actually performed)
DeviceWaitingForEndOfCycle	**device is waiting for end of defined cycle time** This item will get the value '1' when all actions (measurement, shaking, injection) of a cycle are completed but the defined cycle time is not yet over. At this moment there are already measurement values for this cycle available (The item ActCycle shows the number of the current cycle.) but the next cycle has not yet started. This only occurs in plate mode when the user has defined a cycle time which is longer than the required minimum cycle time. This item is therefore useful if you want to show the current cycle or the remaining cycle time in your program.
DevicePausing	**device in pause state** - '0' : instrument is not in pause state - '1' : instrument is in pause state (test run pausing until continue command)
DevicePausingTime	**device in pause time state** - '0' : instrument is not in pause time state - '1' : instrument is in pause time state (test run pausing for the defined time)
DeviceError	**device error** - '0' : no error - '1' : device hardware error

Item	Information provided
DeviceWarning	**device warning** - '0' : no warning - '1' : device warning (e.g. requested timing not met)
Cmdrefused	**command refused** - '0' : command accepted (will be executed afterwards) - '1' : command refused (e.g. wrong command / transmission error)
QuitCode	**instrument quit code** Contains an error code >'0' when command refused
QuitCode2, QuitCode3, QuitCode4, QuitCode5	**additional instrument quit codes** contain additional error information (e.g. volume group, kinetic window)
Error	**last error / warning message** This item will contain the last error message until it will be reset using the ResetError command.
SoftNum	**software version** (version of the Omega server software)
EPROMNum	**firmware version** of the reader e.g.: 'O01101' = V1.10 P1
BoardNum	**main board version / measurement board version** e.g. '1/3'
ReaderType	**type of reader used *** e.g. 'POLARstar Omega' or 'NEPHELOstar Plus' * available beginning with software version 3.00
PlateOut	**position plate carrier** - '0' : plate carrier is inside the instrument - '1' : plate carrier is outside the instrument
ReagOpen	**position reagent door** - '0' : reagent door is closed - '1' : reagent door is open
MeasPlateInserted	**microplate in plate carrier *** - '0' : there is no microplate in the plate carrier - '1' : microplate inserted * This item contains only valid information if the reader microplate sensor has been switched on (reader EEPROM parameter). The status of this item will be updated after a PlateIn, Gain adjustment or Run command.
Pump1In	**pump 1 built in** - '0' : pump 1 is not built in - '1' : pump 1 is built in
Pump2In	**pump 2 built in** - '0' : pump 2 is not built in - '1' : pump 2 is built in
Chan2BuiltIn	**channel 2 built in** - '0' : second fluorescence measurement channel (PMT) not built in - '1' : second fluorescence measurement channel (PMT) built in (necessary for fluorescence polarization and dual emission mode measurements)

Item	Information provided
AbsOptionIn	**absorbance option built in** - ‘0’ : absorbance option is not built in - ‘1’ : absorbance option is built in
LumiOptionIn	**luminescence option built in** - ‘0’ : luminescence option is not built in - ‘1’ : luminescence option is built in
IncubIn	**incubator built in** - ‘0’ : incubator is not built in - ‘1’ : incubator is built in
ExtIncubator	**extended incubator built in** This incubator has an extended temperature range of 10°C to 60°C. - '0' : No extended incubator is built in - '1' : Extended incubator is built in
Temp1	**current temperature of incubator** (**bottom** heating plate) - ‘nn.n’ : temperature in °C
Temp2	**current temperature of incubator** (**top** heating plate) - ‘nn.n’ : temperature in °C
Temp1Miss	**signal temperature 1 missing** - ‘0’ : signal of temperature sensor 1 OK - ‘1’ : signal of temperature sensor 1 not present (no measurement)
Temp2Miss	**signal temperature 2 missing** - ‘0’ : signal of temperature sensor 2 OK - ‘1’ : signal of temperature sensor 2 not present (no measurement)
Temp1UpperLim	**temperature 1 > upper limit** - ‘0’ : temperature 1 OK - ‘1’ : temperature 1 exceeded target value
Temp2UpperLim	**temperature 2 > upper limit** - ‘0’ : temperature 2 OK - ‘1’ : temperature 2 exceeded target value
T1notreached	**temperature 1 not reached** - ‘0’ : temperature 1 reached target value - ‘1’ : temperature 1 hasn't reach target value yet
T2notreached	**temperature 2 not reached** - ‘0’ : temperature 2 reached target value - ‘1’ : temperature 2 hasn't reach target value yet
TestDur	**test run duration in seconds** - ‘n...nnnnn’
IntTime	**interval time in seconds** - ‘nn.n’ : duration of one kinetic interval (only in well mode tests)
Valcount	**number of blocks** - ‘nnnn’: number of data blocks that can be stored in the instrument
MeasureData	**measurement data ready** - ‘0’ : there is no measurement data available - ‘1’ : there is measurement data available from the last test run

Item	Information provided
ActRow ActCol	**row of last well that was measured**: ‘1’ ... ‘16’ **column of last well that was measured**: ‘1’ ... ‘24’ These two values show the last well whose measurement results are available from the instrument (for well mode tests).
ActCycle	**cycle that was measured last**: ‘1’ ... ‘200’ The last kinetic cycle whose measurement results are available in the instrument (for plate mode tests).
ActRowRet ActColRet	**row of last well that was read back**: ‘1’ ... ‘16’ **column of last well that was read back**: ‘1’ ... ‘24’ These two values show the last well whose measurement results are already transferred to the pc (for well mode tests).
ActCycleRet	**cycle that was read back last**: ‘1’ ... ‘200’ The last kinetic cycle whose measurement results are already transferred to the pc (for plate mode tests).
OffsetData	**offset data ready** - ‘0’ : data from offset determination is not available - ‘1’ : data from offset determination is available
TimeData	**time data ready** - ‘0’ : the test run / cycle time is not available - ‘1’ : the test run / cycle time for the last test run is available
GainData	**gain data ready** - ‘0’ : gain values are not available - ‘1’ : gain values from the last gain adjust procedure are available
Gain1	**gain value for measurement channel 1** that was determined by ‘GainWell’ or ‘GainPlate’ command - ‘0’...‘4095’
Gain2	**gain value for measurement channel 2** that was determined by ‘GainWell’ or ‘GainPlate’ command - ‘0’...‘4095’
Gain1Raw	**obtained raw result absolute during gain adjustment for measurement channel 1** - ‘0’...‘260 000’ (fluorescence mode) - ‘0’...‘100 000’ (luminescence mode)
Gain2Raw	**obtained raw result absolute during gain adjustment for measurement channel 2** - ‘0’...‘260 000’ (fluorescence mode) - ‘0’...‘100 000’ (luminescence mode)
Gain1Percent	**obtained raw result in % of meas. range during gain adjustment for measurement channel 1** - ‘0’...‘100’ %
Gain2Percent	**obtained raw result in % of meas. range during gain adjustment for measurement channel 2** - ‘0’...‘100’ %
KFactor	**value for K factor**, that was determined by ‘GetKFactor’ command - ‘-9.99’...‘99.99’
ActRowGain, ActColGain	**row and column of well with was used for gain adjustment** (well with highest value for GainPlate)

Item	Information provided
MotorEnabled	**motors enabled** - '0' : motors disabled - '1' : motors enabled
StackerStatus	**current stacker status** only available, if a stacker is attached to the reader - 'Ready': stacker is in standby state - 'Busy': stacker is busy In case of an error you will find here the error message.
StackerKindOfResponse	**type of last response from stacker:** - 'confirmation response' (confirmation of a received command) - 'barcode data response' - 'offset data response' - 'EEPROM data response' - 'system test data response' - 'CPU port data response' - 'barcode scanner parameter response' - 'firmware data response'

An item '**Terminate**' also exists. When the Omega software is terminated (for example, by the user), this item gets the value 'TERMINATE'. You can use this item to shut down your client software automatically when the server software is terminated.

3 Control Commands

Each command is a macro in terms of DDE interface. It consists of a **macro name** (of string type) and some **parameters** (also of string type).

Here is the list of the possible commands with the corresponding parameters. All strings are **not case sensitive**.

3.1 Dummy

This command does nothing. It can be used to check the ActiveX or DDE connection. There are no parameters.

Remark: This command is always allowed.

3.2 Init

This command initialize the reader. There are no parameters.

Remark: This command is always allowed.

3.3 User

Using this command, you can send login information via the ActiveX or DDE connection.

Parameter		Meaning
1.	**user name**	name of user as defined in the Omega user database
2.	**data path**	absolute path that points to a directory where the measurement data of runs performed by this user should be stored (where the database files `measure.dbf` and `<number>.dbf` will be found) This data path is usually: <root directory>\data
3.	**root directory**	Specify root directory for this user (usually ~:\Program Files\BMG\Omega\<username>). The test definitions for this user will be searched under: <root directory>\definit
4.	**run only**	If this parameter is 'True' (or '1') the user will be logged in without permission to change test protocols.

Remark: This command is always allowed.

3.4 PlateIn / PlateOut

These commands are used to move the plate carrier into the instrument or out of the instrument.

‘PlateIn’ Moves the plate carrier into the instrument.

‘PlateOut’ Moves the plate carrier out of the instrument.

Parameter		Meaning
1.	**mode**	‘Normal’: normal plate in/out movement ‘Right’: moves plate carrier out, whereby the plate carrier will move to the right hand side (for special robotic requirements, option is only available for 'PlateOut') ‘User’: user defined plate movement: With **X and Y** position for ‘PlateOut’ and ‘PlateIn’
2.	**X coordinate**	for ‘PlateIn’ and Mode ‘User’ between -25 and 3810 for ‘PlateOut’ and Mode ‘User’ between -25 and 3250 / 3270 * * X values in the range 3251 … 3270 are only allowed if the Y value is 4060 or higher
3.	**Y coordinate**	for ‘PlateIn’ and Mode ‘User’ between -190 and 1590 for ‘PlateOut’ and Mode ‘User’ between 3840 and 4280 * * when a stacker is attached to the reader only the range 4210 to 4280 can be used

If the plate carrier is already in the requested position, the instrument gives a positive response and takes no action.

Remark: This command is only allowed when

- the instrument is in standby state

3.5 Pump1 / Pump2

These commands are used to prime (prime / backflush direction) or initialize the injectors.

Parameter		Meaning
1.	**number of strokes**	Number of strokes [1...9] defines the number of strokes that are performed for the prime / backflush procedure (1 stroke = 500 µl).
2.	**pump speed**	Pump speed [1...12] defines the dispensing speed when the pump is active. 1: maximum speed (420 µl/s) 12: minimum speed (100 µl/s)
3.	**direction**	Direction determines the direction of the pump movement. 0: Prime : Liquid runs to the injection needle 1: Backflush : Liquid runs to the reagent bottle
4.	**invert dispensing function**	Invert dispensing function determines the order of the plunger movements (dispense and aspirate) when injecting and also the zero position of the plunger when no action is taking place. 0: The dispensing function is set to standard mode, which means that the zero position of the plunger is at the bottom position. When injecting, the dispensing action will take place before the aspirating action. This should be the normal setting because in well mode the reaction in the chemistry starts at the predefined pump start time. 1: The dispensing function is set to inverted mode which means that the zero position of the plunger is at the top position. When injecting, the dispensing action will take place after the aspirating action. It has to be considered that in well mode the reaction of the chemistry is delayed after the predefined pump start time, because the dispensing action takes place after the aspirating action. This mode is especially important when there are cells in the liquid. Since the plunger stays in the top position when no action is taking place, it is not possible that the cells descend to the bottom of the cylinder. So during injection the cells will be transported to the needle and will not stay in the cylinder.

The first ‘Pump1’ / ‘Pump2’ command will initialize the selected dispenser before the priming procedure itself takes place. Subsequent ‘Pump1’ / ‘Pump2’ commands will dispense and aspirate syringe volumes for a defined number of times (n). For the direction of the priming procedure, choose between prime (to fill the liquid system → liquid runs to the dispensing needle) and backflush (to empty the liquid system → liquid runs to the reagent bottle).

The dispensing speed can be changed between 1 (maximum speed) and 12 (minimum speed). The pickup speed is always slower than dispensing speed by a factor of 1.5.

!! A T T E N T I O N !!

When the Prime mode is selected, the injection needle must be removed from the measurement head, since liquid will run out of the needle. A warning in the client software is useful to prevent contamination of the instrument.

The operator assumes all responsibility when priming the dispenser !!

Remark: This command is only allowed when

- the instrument is in standby state
- the selected dispenser is built in the instrument

3.6 Temp

This command is used to set the target temperature of the incubator unit in 0.1°C increments, and to switch on or to switch off the incubator.

Parameter		Meaning	
1.	**nominal temperature**	'00.0':	The incubator unit will be switched off.
		'00.1':	The temperature will not be controlled (no heating), but the temperature will be measured ("temperature monitoring").
		'25.0'...'45.0':	The incubator will be switched on and the new target value will be set. The value can be changed in steps of 0.1 °C.
		'10.0'...'60.0':	This range is only available with the extended incubator. The incubator will be switched on and the new target value will be set.

When using tests that require incubation, the operator typically checks the temperature and decides when the test run can be started or whether the test run must be stopped due to a temperature difference.

If there is an error with a temperature sensor or a heating element, the incubator will be automatically switched off and a temperature error will be reported. But apart from this, the instrument will work correctly and all further commands are allowed. With a new 'Temp' command (nominal temperature = 0) the temperature error can be reset.

Remark: This command is only allowed when

- the instrument is in standby state
- the incubator is built in

3.7 GainWell / GainPlate / GetKFactor

The '**GainWell**' command automatically calculates the optimum sensitivity setting (amplification for PMT), based on the measurement values of a single well measured at different gain values. When using a dual emission measurement mode (fluorescence polarization, dual emission fluorescence, dual emission luminescence) the gain values for both channels will be determined simultaneously. When you use multichromatics you need to send a separate gain command for each used chromatics (each used optic module).

In addition, this command can be used to perform a focus adjustment.

The '**GainPlate**' command automatically calculates the optimum sensitivity setting (amplification for PMT), based on the measurement values of the well with the highest result that is measured at different gain values. Before the gain adjustment itself takes place all the measurement points which are defined in the layout will be measured to find the well with the maximum result.

The '**GetKFactor**' command calibrates the ratio between the two measurement channels without changing the gain values (= a kind of fine adjustment, only important for fluorescence polarization test runs, see below).

Parameter		Meaning	
1.	**protocol name**	name of a particular test as defined in the Omega program (test definition library)	
2.	**path to protocol definition**	absolute path that points to a directory where the test run definitions are stored (where the database file `testdef.db` is to be found)	
3.	**column of well**	Specify well which should be used for test measurements (only for 'GainWell' and 'GetKFactor').	
4.	**row of well**	Specify well which should be used for test measurements (only for 'GainWell' and 'GetKFactor').	
5.	**required value channel A**	'0'...'100'	desired raw result for first measurement channel (channel A) in % of measurement range *
6.	**required value channel B**	'0'...'100'	desired raw result for second measurement channel (channel B) in % of measurement range – only for fluorescence polarization or dual emission protocols of importance *
7.	**filter setting number**	'1'...'8'	Choose the filter setting (chromatic) to be used. If the test protocol uses more than one chromatic (multichromatic tests) you should perform a gain adjustment for each chromatic (each used filter setting).
8.	**target polarization value**	'0'...'500'	Only used for 'GainWell' and 'GetKFactor' for fluorescence polarization test protocols: Target fluorescence polarization value for the sample with known polarization in mP.

Remark: This command is only allowed when

- the instrument is in standby state

* You can also use non-integer values for these two parameters, e.g. '75.54'.

3.7.1 Gain Adjustment in Fluorescence Polarization Mode

You should perform a GainWell command to adjust the gain value for both channels. Please specify a well with know fluorescence polarization value and enter this known target mP value as parameter number 8. If a fluorescein based chemistry will be used the target polarization value should be set to 35. After performing the gain command in Fluorescence Polarization mode a K-Factor for fine-adjustment of the two measurement channels will automatically be calculated (based on the obtained raw results and the used target mP parameter). Therefore, it is no necessary to send a separate GetKFactor command after performing the gain adjustment. The K-Factor value will be used to calculate the polarization values for all wells of the test run.

Instead of specifying the required values for both channels you can also only specify a value for channel A. When entering '0' for channel B the required value for this channel necessary to achieve a K-Factor as close to 1 as possible will be automatically calculated based on the required value for channel A and the target mP parameter.

It is possible to use the GetKFactor command to determine a new K-Factor without changing the gain values.

Gain adjustment over the whole plate (GainPlate) is not available for fluorescence polarization protocols.

3.8 SetGain

The ‘SetGain’ command can be used to change the gain values defined as part of a test protocol.

Parameter		Meaning	
1.	**protocol name**	name of a particular test protocol as defined in the Omega server program (test definition library)	
2.	**path to protocol definition**	absolute path that points to a directory where the test run definitions are stored (where the database file `testdef.db` is to be found)	
3.	**filter setting number**	‘1’...‘8’	Choose the filter setting (chromatic) to be used. If the test protocol uses more than one chromatic (multichromatic tests) you should perform a gain adjustment for each chromatic (each used filter setting).
4.	**channel**	‘A’ or ‘1’	change the gain value for channel A
		‘B’ or ‘2’	change the gain value for channel B (for dual emission or FP protocols)
5.	**gain value**	‘0’...‘4096’	new gain value

3.9 SetSampleIDs / ClearSampleIDs / ClearDilutionFactors

The ‘**SetSampleIDs**’ command will read sample ID values from a file and store these IDs inside the specified protocol. This command should be used if necessary before starting the measurement using the ‘Run’ command. The sample ID file can be an ASCII or Excel (XLS) format file. The file content should use the same syntax as for the 'Import IDs' function of the Sample IDs sheet of the ‘Start Measurement’ dialogue of the Omega software (third sheet). You can use this dialogue to create the sample IDs file.

If the sample IDs file contains dilution factors these factors will also be imported. If the file contains entries for wells, which are not used in the selected test protocol, these entries will be ignored (e.g. well A13 for a protocol using a 96 well plate). If the sample IDs file contains invalid entries, e.g. an impossible well name like A49 or XYZ, there will be an error message.

Use the ‘**ClearSampleIDs**’ command to delete the Sample IDs associate with a test protocol.

Use the ‘**ClearDilutionFactors**’ command to reset all dilution factors to 1.

Parameter		Meaning
1.	**protocol name**	name of a particular test protocol as defined in the Omega server program (test definition library)
2.	**path to protocol definition**	absolute path that points to a directory where the test run definitions are stored (where the database file `testdef.db` is to be found)
3.	**Sample IDs file name**	only for SetSampleIDs: file name including path of the file containing the Sample IDs

3.10 Run / CalculateTestDuration

The 'Run' command can be used to initiate a measurement.

The 'CalculateTestDuration' command can be used to determine how long a test run will last without actually starting the measurement. The result will be available via the item 'TestDur'.

Parameter		Meaning
1.	**protocol name**	name of a particular test protocol as defined in the Omega control program (test definition library)
2.	**path to protocol definition**	absolute path that points to a directory where the test run definitions are stored (where the database file `testdef.db` is to be found) *
3.	**path for measurement data**	absolute path that points to a directory where the measurement results will be stored
4.	**plate ID1**	plate identifier for current test run (max. 100 chars)
5.	**plate ID2**	plate identifier for current test run (max. 100 chars)
6.	**plate ID3**	plate identifier for current test run (max. 100 chars)

The plate ID parameters are optional.

Remark: This command is only allowed when

- possible required injectors have already been primed

* The path to the protocol definition for microplate measurements is the user root directory plus '\Definit', e.g. 'C:\Program Files\BMG\Omega\User\Definit'. For using a protocol from the LVis Plate tab replace '\Definit' with '\DefLVis', e.g. use 'C:\Program Files\BMG\Omega\User\DefLVis'.

3.11 Pause

This command can be used to temporarily stop an active plate mode test run before beginning a certain cycle. A pause in a test run may be useful to take out the plate to inject something or to incubate the plate.

Parameter		Meaning	
1.	**cycle**	'1' ... No. of cycles: '65535':	stop measurement run before beginning this cycle use '65535' to get a pause before the next cycle *

Remark: This command is only allowed when

- a plate mode test run is active

* When using a software version before 3.00 or a firmware version before 1.30 use '255' instead of '65535'.

3.12 Continue

This command can be used to continue the test run after a pause. There are no parameters.

Remark: This command is only allowed when

- the reader is in test run pause state
- the plate carrier is inside the reader

3.13 StopTest

This command immediately stops an active test run, whereby the transport system and all instrument actions will stop immediately.

Parameter		Meaning
1.	**save results**	'Save': store the measurement result data if available 'Nosave': don't store the result data

Remark: This command is only allowed when

- a test run is active or the instrument is in test run pause state

3.14 StopSystem

This command immediately stops any reader activity or any stacker activity controlled by the reader firmware, if a stacker is attached to the reader.

Remark: This command is only allowed when

- a test run is active or the instrument is in test run pause state.

3.15 MotorDis

The 'MotorDis' command can be used to disable (power down) the stepper motors. This is useful if there are no instrument actions for a long time to prevent heat build-up and to save power. Normally it is not necessary to send this command since the motors will be automatically disabled during inactivity by the Omega program. There are no parameters.

Remark: This command is only allowed when

- the instrument is in standby state
- the plate carrier is inside the instrument

3.16 MotorEn

To enable the stepper motors, you can use the 'MotorEn' command. Normally it is not necessary to send this command since the motors will be automatically enabled during a 'Plate..' or 'Run' command. There are no parameters.

Remark: This command is only allowed when

- the instrument is in standby state
- the plate carrier is inside the instrument

3.17 ResetError

The 'ResetError' command changes the actual status of the item 'Status' ('DdeServerStatus') from 'Error' back to 'Ready', 'Busy' or 'Running' depending upon the action being performed and deletes the last error message (item 'Error'). There are no parameters.

Remark: This command is always allowed. The status (but not the last error message availabllbe via the item 'Error') will also automatically be reset after receiving a new command other than Dummy, MotorDis or MotorEn.

3.18 Terminate

This command will shutdown the Omega software. Before exiting, the Omega software will initialize the reader to ensure the plate carrier is inside the instrument and all activities are stopped. There are no parameters.

Remark: This command is always allowed.

4 Summary of Commands

Command	Meaning
Dummy	does nothing (can be used to check the ActiveX or DDE connection)
CalculateTestDuration	calculates the duration of a test run
ClearDilutionFactors	reset all dilution factors of a test protocol to 1
ClearSampleIDs	deletes the Sample IDs associate with a test protocol
Continue	continues a temporarily stopped test run
GainPlate	determines optimum gain value on well with maximum raw result
GainWell	determines optimum gain value on specified well
GetKFactor	determines K-Factor value
Init	initializes the reader (FLUOstar Omega, LUMIstar Omega, POLARstar Omega, SPECTROstar Omega or NEPHELOstar Plus)
MotorDis	disables all stepper motors
MotorEn	enables stepper motors
Pause	temporarily stops an active plate mode test run
Pump1	initializes / primes dispenser system 1
Pump2	initializes / primes dispenser system 2
PlateIn	moves plate carrier into the instrument
PlateOut	moves plate carrier out of the instrument
ResetError	resets error status
Run	starts a test run
SetGain	changes the gain value(s) of a test protocol
SetSampleIDs	read sample ID values from a file and store these IDs inside the specified protocol
StopTest	aborts performing test run
StopSystem	stops all reader / stacker activities
Temp	sets target temperature and switches incubator on/off
Terminate	terminates the Omega software
User	sends user information to the Omega software

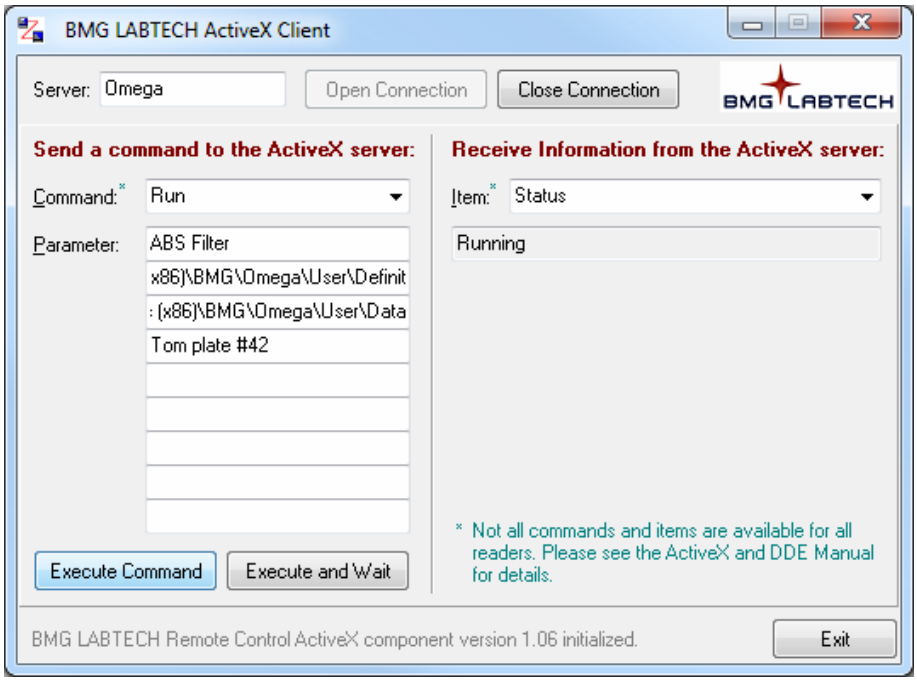
5 Example Client Programs

The example programs ActiveXClient.exe and DdeClient.exe, also installed with the Omega software package and found in the subdirectory \CLN of the Omega main directory (usually c:\Program Files\BMG\Omega\CLN), are offered as an illustration for applying the ActiveX or DDE interfaces described previously.

5.1 ActiveX Client Example Program

You can use all BMG LABTECH reader control programs installed on your computer as ActiveX server (see chapter 1.2 for server names). After entering the server name click the 'Open Connection' button. This will start the selected server and initialize the reader.

The program allows ActiveX **commands** to be sent to the ActiveX server. You can select a command from the pull down list on the left side or you can enter its name manually. Do not forget to enter the necessary parameters.

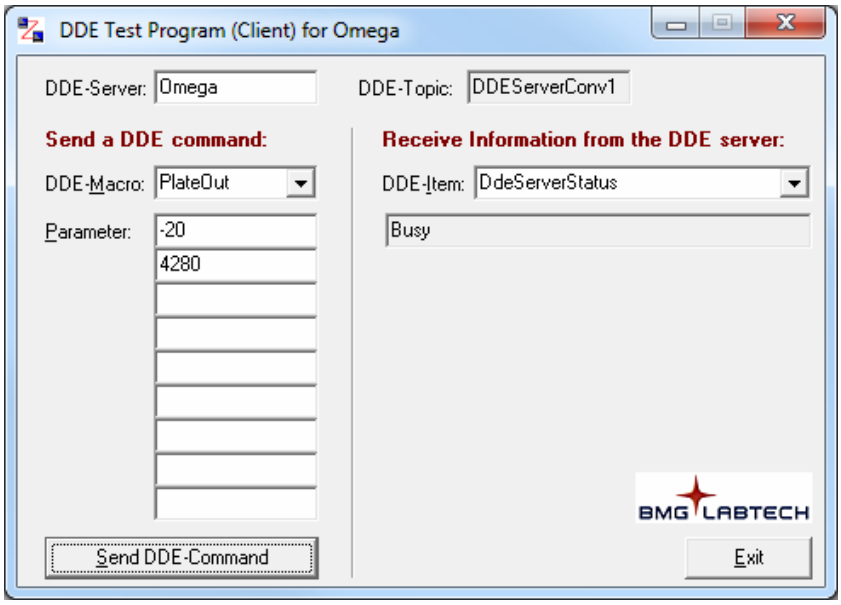


You can select an information **Item** on the right side of the program window. Its value will be shown in the box below the Item pull down box.

Note: This example client program will update the value of the selected item continuously. Due to technical reasons there will be no update while executing a command using the 'Execute and Wait' method.

5.2 DDE Client Example Program

The program allows **DDE macro commands** to be sent to DDE-server. You can select a command from the pull down list on the left side or you can enter its name manually. Do not forget to enter the necessary parameters.



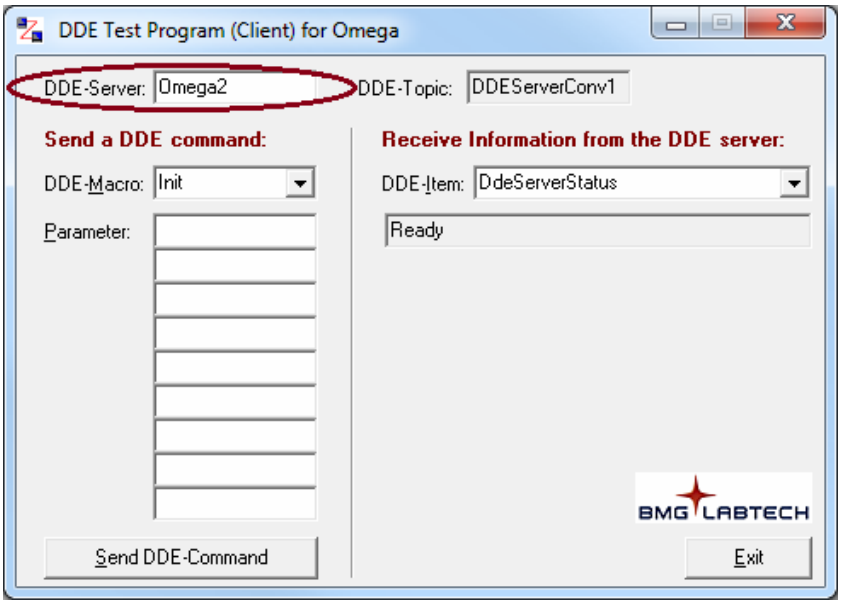
You can select a **DDE Item** on the right side of the program window. Its value will be shown in the box below the DDE Item pull down box.

6 Using Multiple Program Installations

It is possible to install the Omega control program more than once. You can create up to 9 program installations (see software manual). This will be useful if more than one Omega reader is connected to one computer.

The ***ActiveX*** *or* ***DDE server name*** for the first installation is ***Omega***. The server name for the second installation is ***Omega2***, for the third ***Omega3*** and so on.

There will be one ActiveX client example program and one DDE client example program (see chapter 5) installed with every program installation. You will find these programs in the sub directory '`\Cln`' of the main installation directory, e.g. '`c:\Program Files\BMG\Omega2\Cln`'. You can use the client example programs from the different installations simultaneously. In addition, it is possible to change the used server from inside the client example program.

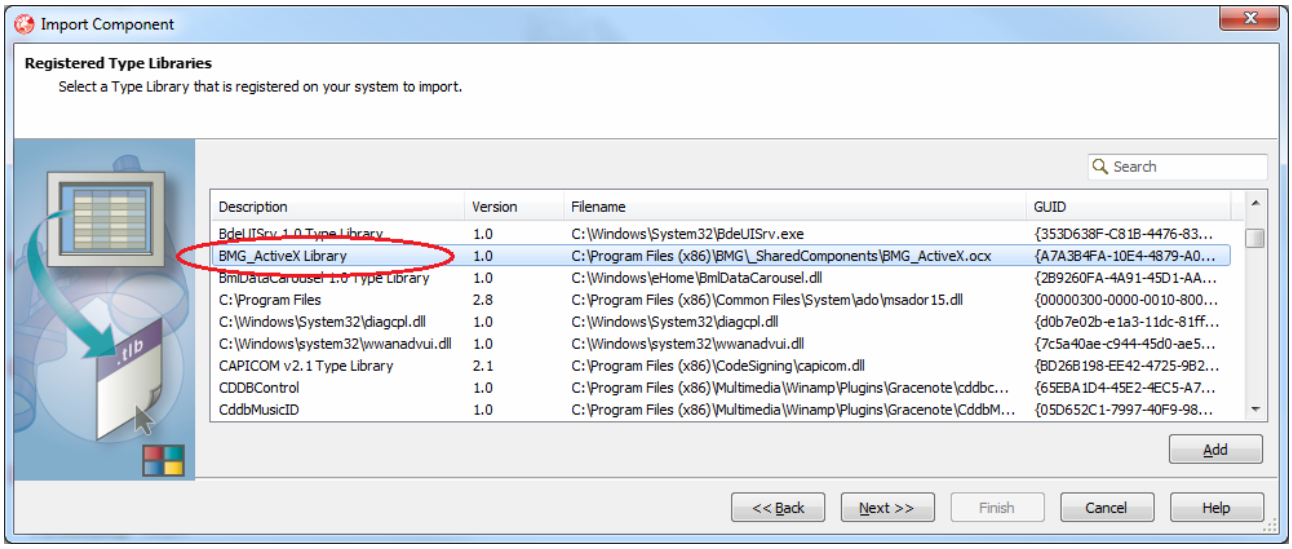


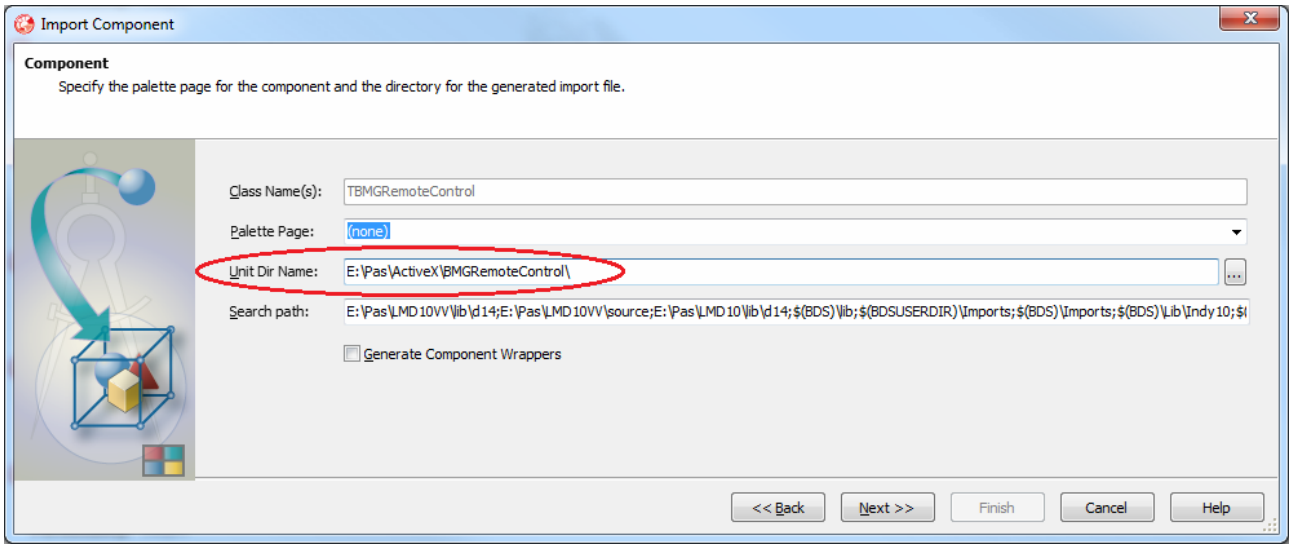
7 Delphi Programming Examples

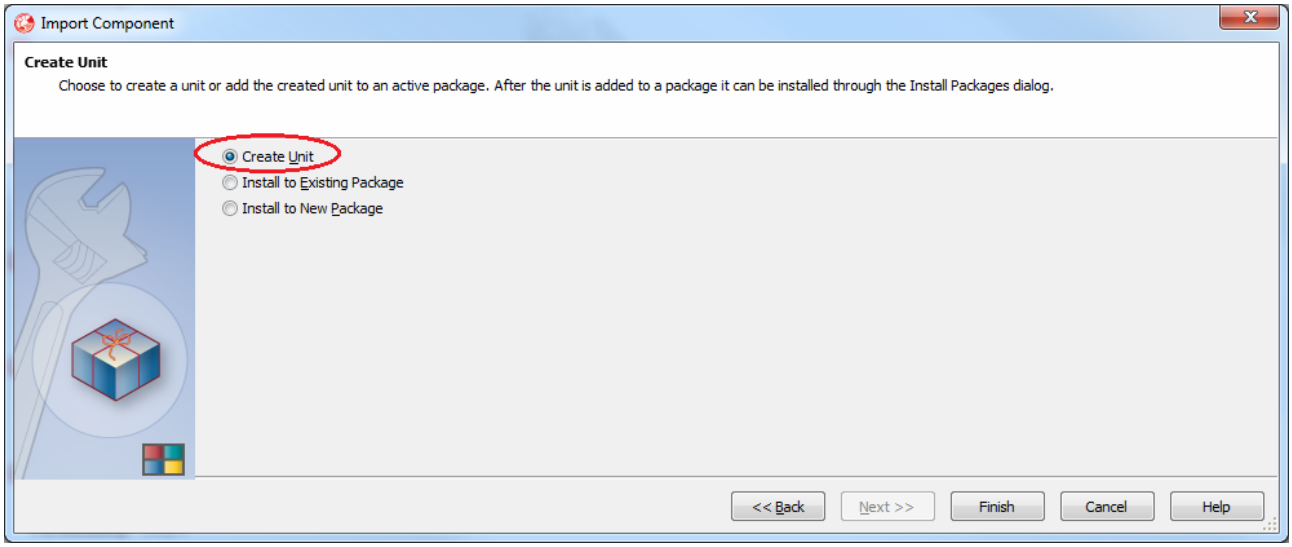
7.1 Using the ActiveX Component with Delphi

7.1.1 Importing the Type Library

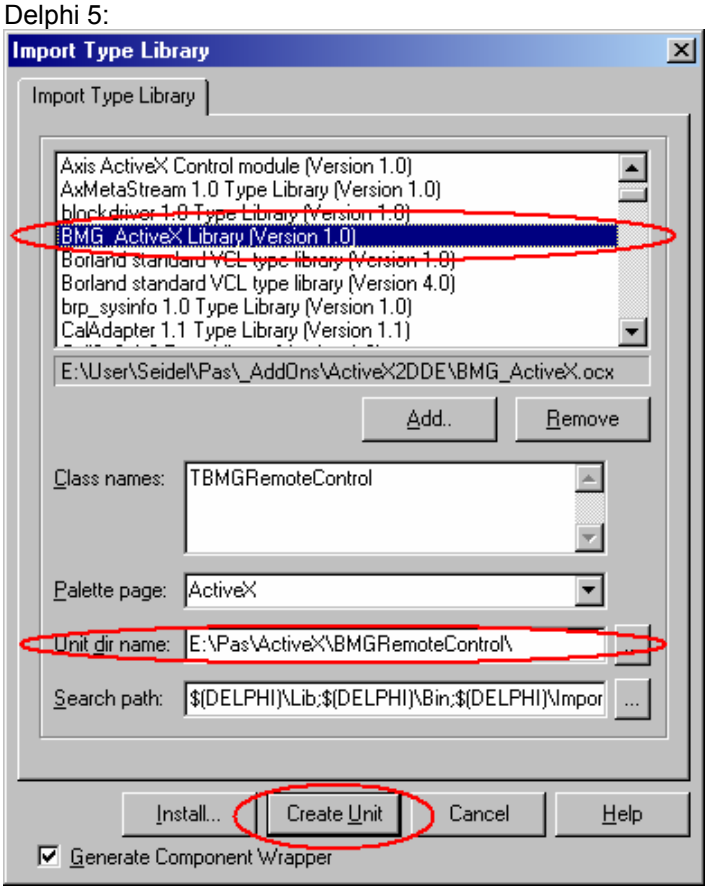
To use the ActiveX component a unit containing the Pascal version of the interface (“type library”) is necessary. Such a unit can be created by Delphi automatically. Use the command ‘Component | Import Component | Import Type Library’ (Delphi 5: ‘Project | Import Type Library’). Select the BMG_ActiveX Library. Enter the target directory for this unit (*Unit dir name*) and click the ‘Create unit’ button.







Delphi 5:



This will create a unit BMG_ActiveX_TLB and will include the unit in your project.

7.1.2 Opening the Connection

Define one instance:

```
type
  TClientform = class(TForm)
    ...
  private
    fBMGRemoteControl : iBMGRemoteControl;
    BMGRemoteControlActiveXinitialized : boolean;
  end;
```

Initialize the ActiveX control instance and open a connection using the selected ServerName:

```
procedure TClientform.OpenConnection;
var
  st  : string;
  res : OleVariant;
begin
  if Not(BMGRemoteControlActiveXinitialized) then begin
    fBMGRemoteControl := CoBMGRemoteControl.Create;
    BMGRemoteControlActiveXinitialized:=true;
  end;

  fBMGRemoteControl.OpenConnection(PChar(Servername),res);
  if res=-2 then  {Error}
    ShowMessage('An ActiveX Server "'+Servername +'" is not registered!')
  else
  if res=-1 then  {Warning}
    ShowMessage('Connection to "'+Servername +'" is already active!');
end;
```

7.1.3 Sending a Command to the ActiveX Server

```
procedure TClientform.btExecuteClick(Sender: TObject);
var
  v, res : OleVariant;
begin
  v:=VarArrayCreate([0,9], varOleStr);
  v[0]:='Run';                                          {Add command}
  v[1]:='FI PROTOCOL 1';                                {Add 1. parameter}
  v[2]:='C:\Program Files\BMG\Omega\User\Definit';       {Add 2. parameter}
  v[3]:='C:\Program Files\BMG\Omega\User\Data';          {Add 3. parameter}
  v[4]:='Batch: 08';                                    {Add 4. parameter}
  v[5]:='Plate: 15';                                    {Add 5. parameter}
  v[6]:='SN: 415-0042';                                 {Add 6. parameter}
  v[7]:='';                                             {Add 7. parameter}
  v[8]:='';                                             {Add 8. parameter}
  v[9]:='';                                             {Add 9. parameter}

  fBMGRemoteControl.Execute(v, res);

  if res=-1 then
    MessageDlg('Connection to ActiveX server is not open!', mtError, [mbOk], 0)
  else
  if res<-1 then
    MessageDlg('Sending the command failed (error code '+IntToStr(_Result)+')!',
               mtError, [mbOk], 0);
end;
```

The function **Execute** will return immediately after sending the command. You can now use GetInfo('Status',res) to wait until the processing of the command has been finished (e.g. 'Ready' → 'Busy' → 'Ready').

If you use the function **ExecuteAndWait** instead of Execute, the function will not return before processing the command has been completed. It might be a good idea to change the cursor before and after calling ExecuteAndWait to give the user some feedback:

```
  Screen.Cursor:=crHourGlass;
  fBMGRemoteControl.ExecuteAndWait(v, res);
  Screen.Cursor:=crDefault;
```

7.1.4 Retrieving Information from the ActiveX Server

You can use a timer to get the value of the selected item updated automatically.

```
procedure TClientform.GetInfoTimerTimer(Sender: TObject);
var
  v : OleVariant;
begin
  fBMGRemoteControl.GetInfo(PChar(ItemEdit.text),v);
  edAnswer.text:=v;
end;
```

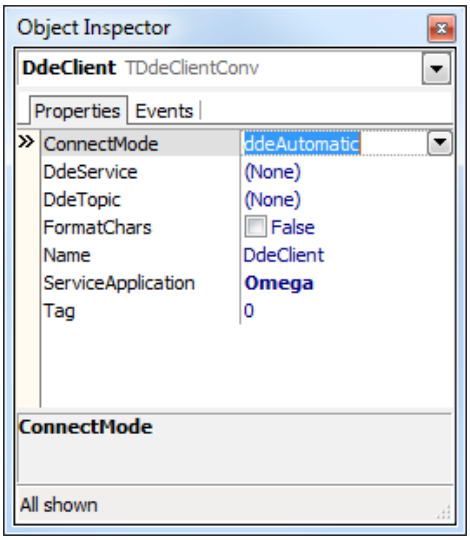
7.1.5 Closing the Connection

```
procedure TClientform.CloseConnection;
begin
  fBMGRemoteControl.CloseConnection;
end;
```

7.2 Using the DDE Interface with Delphi

7.2.1 TDdeClientConv Component

Insert a TDdeClientConv component in a form. Select ConnectMode=ddeAutomatic using Delphi´s object inspector.



7.2.2 Find the DDE Server Program

Write a procedure to get the path to the newest Omega program file from the windows registry:

```
procedure GetOmegaPathFromRegistry;
var
  RegIniFile          : TRegIniFile;
  VersionsKeys        : TStrings;
  OmegaPath,
  NewestVersion       : string;
  i                   : integer;
begin
  RegIniFile:=TRegIniFile.Create('');
  RegIniFile.RootKey:=HKEY_LOCAL_MACHINE;
  RegIniFile.OpenKey('\SOFTWARE\BMG Labtechnologies\Omega',false);

  {find out which Omega versions are installed:}
  VersionsKeys:=TStringList.Create;
  RegIniFile.ReadSections(VersionsKeys);

  {and now get the newest: (this simple string compare routine will only work
   until version 9.9!)}
  NewestVersion:='0';
  for i:=0 to VersionsKeys.count-1 do
    if VersionsKeys[i]>NewestVersion then NewestVersion:=VersionsKeys[i];

  {read the path to the newest Omega software:}
  OmegaPath:=RegIniFile.ReadString('\SOFTWARE\BMG Labtechnologies\Omega\'+
                                   NewestVersion+'\ControlApp',
                                   'MainDir','***');

  {set the DDE service application name:}
  if pos('***', OmegaPath)>0 then {nothing found? - then use only exename:}
    DdeClient.ServiceApplication:='Omega'
  else
    DdeClient.ServiceApplication:=OmegaPath+'\EXEDLL\Omega';

  VersionsKeys.Free;
  RegIniFile.Free;
end; {GetOmegaPathFromRegistry}
```

7.2.3 Start DDE Connection

```
GetOmegaPathFromRegistry;
if not(DDEClient.SetLink ('Omega','DDEServerConv1')) then
  ...  {some kind of error message}
```

7.2.4 Send a Command to the DDE Server

In Delphi there are two DDE send commands. The first one is for DDE macro commands without parameters:

```
DDEClient.ExecuteMacro('Init', false);
```

If you have to send a macro command with parameters use the second form:

```
var
  cmd : TStrings;

begin
  Cmd:=TStringList.Create;      {create stringlist}
  Cmd.Add('PlateOut');          {add DDE macro command name}
  Cmd.Add('User');              {add parameters}
  Cmd.Add('0');
  Cmd.Add('5000');

  {send command to DDE server:}
  if not(DDEClient.ExecuteMacroLines(Cmd, false)) then
    ...  {try to reopen DDE link or some kind of error message}

  Cmd.free;                     {release stringlist}
end;
```

7.2.5 Retrieve Information from the DDE Server

There are two ways to retrieve information from the DDE server in Delphi:

1. Use a TDdeClientItem component

Use, for example, a TDdeClientItem component with the name DdeClientTerminate.

After opening the DDE connection set the DdeItem-value of this component to one of the DDE items available from the Omega DDE server.

```
DdeClientTerminate.DdeItem:='Terminate';
```

Implement a procedure for the OnChange event of this component. This procedure will be called automatically if the value of the DDE item has changed:

```
procedure TDDE_Client.DdeClientTerminateChange(Sender: TObject);
begin
  {Shutdown client program when the user exits the Omega program:}
  if AnsiUpperCase (copy (DdeClientTerminate.Text,1,9))='TERMINATE' then
    ExitProgram(Sender);
end;
```

2. Use the RequestData command

Use a timer and ask the server for the values you are interested in:

```
St:=DdeClient.RequestData ('Status');
```

8 Visual Basic Programming Example

8.1 Using the ActiveX Component with Visual Basic

Programming an ActiveX client application in visual basic is similar to programming it in Delphi. It is not necessary to create something like an interface unit, but the component needs to be registered (which will be done by the BMG LABTECH installation program).

8.1.1 Opening the connection:

```
Dim activex As BMGRemoteControl

Sub OpenActiveXConnection()
    Dim res As Variant
    Set activex = New BMGRemoteControl

    activex.OpenConnection ("Omega"), res
    MsgBox res
End Sub
```

8.1.2 Sending a Command to the ActiveX Server:

```
Sub ExecuteCmd()
    Dim res As Variant
    Dim CmdAndParameter As Variant
    ReDim CmdAndParameter(10)

    CmdAndParameter(0) = "PlateOut"
    CmdAndParameter(1) = "Normal"

    activex.Execute CmdAndParameter, res

    MsgBox res
End Sub
```

8.1.3 Retrieving Information from the ActiveX Server

```
Sub RetrieveInfo()
    Dim res As Variant

    activex.GetInfo "Status", res

    MsgBox res
End Sub
```

8.1.4 Closing the connection:

```
Sub CloseActiveXConnection()
    activex.CloseConnection
End Sub
```

8.2 Using the DDE Interface with Visual Basic

Programming a DDE client application in visual basic is similar to programming it in Delphi. In visual basic there is only one DDE macro send command (LinkExecute). For sending a DDE macro with parameters in Delphi, you can use a variable of type stringlist. Unfortunately, the stringlist type does not exist in visual basic. In visual basic, you have to use a simple string variable where you combine the command name and the parameters using the CR/LF chars as separator.

```
Private Sub Command1_Click()
  Dim Cmd, Separator

  ' open DDE connection:
  If Text1.LinkMode = vbLinkNone Then
    Z = Shell("D:\Programme\BMG\Omega\ExeDLL\Omega", 4)
                                                    ' start DDE Server
    Text1.LinkTopic = "Omega|DDEServerConv1"        ' set DDE topic
    Text1.LinkItem = "DdeServerStatus"              ' set DDE item
    Text1.LinkMode = vbLinkManual                   ' choose link mode
  End If

  ' send a DDE command:
  Separator = Chr(13) & Chr(10)

  Cmd = "PlateOut"                                  ' Create command string
  Cmd = Cmd & Separator & "User"                    ' Add parameters
  Cmd = Cmd & Separator & "0"
  Cmd = Cmd & Separator & "5000"

  Text1.LinkExecute Cmd                             ' send DDE macro command

  MsgBox "DDE command to Omega sent.", 64
End Sub
```

9 Using DDEclient as Interface

If you do not want to program an ActiveX or DDE client by yourself, it is also possible to use the program DDEclient (delivered together with the Omega software, see chapter 5) as an interface.

In this operating mode you will call the DDEclient program with command line parameters. DDEclient will interpret these command line parameters as a DDE command including parameters, will start the Omega software (if not already started), establish a DDE connection and send the command via DDE. It will wait until processing of the command has been finished and finish itself then.

You can use all commands described in chapter 3. If there is an error, e.g. a non existing test protocol has been specified for the Run command, a message box will pop up if you do not use the option /e (see below).

9.1 Options

/e If you include this option in the command line no error message box will appear after an error occurred. The error message will be stored only in the Windows registry (see below).

/v If you include this option in the command line the DDEclient program window will be opened. This is mainly useful for test purposes.

9.2 Exit Code

The return value (exit code) will be 0 if the command has been processed successfully.

If the command was rejected by the reader (e.g. due to a parameter range error), the return value will be in the range 1 to 999. *

If the command was rejected by the Omega software (e.g. due to specifying a non existing protocol name), the return value will be 1000. *

If there was no communication to the reader possible (communication time out occurred), the return value will be 2000.

A syntax error in the program call (e.g. missing DDE command by only specifying an option like ‘DDEclient /e’) will result in an exit code of 9999.

* A detailed error message will be available via the Windows registry (see chapter 9.3) and displayed if the option /e was not used.

9.3 Status Information

The current instrument status (see DdeServerStatus in chapter 3) will be stored in the Windows registry under HKEY_CURRENT_USER\Software\BMG Labtechnologies\Omega\1\DDEclient – **Status**. This entry can get the following values: Busy, Ready or Error.

If an error occurred you can get the error message using HKEY_CURRENT_USER\Software\BMG Labtechnologies\Omega\1\DDEclient – **Error**.

9.4 Recommendations

You should call the DDEclient program using a routine, which waits until the program has been finished. Alternatively you can poll the Status value in the registry until this gets ‘Ready’ or ‘Error’.

Use the ‘Dummy’ command to start the Omega software in DDE-Mode. After starting the software the reader will automatically be initialized (which will last a few seconds).

When using the command line operation mode you can omit the path parameters of the Run command. In this case the test protocol and measurement data paths of the standard user “USER” will be used.

To terminate the Omega software use the ‘Terminate’ command.

9.5 Example

DDEclient Dummy
DDEclient PlateOut
… (robot will insert microplate into plate carrier)
DDEclient Run “TOM’S PROTOCOL”
DDEclient PlateOut
… (robot will remove microplate from plate carrier)
DDEclient Terminate

10 How to use Omega Software together with a FLUOstar Galaxy DDE Interface

It is not possible to use the FLUOstar Galaxy software together with a FLUOstar Omega, LUMIstar Omega, POLARstar Omega, SPECTROstar Omega or NEPHELOstar Plus reader. If there is currently only a DDE interface for the FLUOstar Galaxy (or any other BMG LABTECH) software available it is possible to use this interface together with the Omega software due to the fact that the DDE protocols are very similar.

To enable operation of the FLUOstar Galaxy and the Omega software on the same computer the programs use different data base aliases, different registry keys, different directories and a different program name = DDE server name ('Fluo32' for FLUOstar Galaxy and 'Omega' for the Omega software).

To use the Omega software together with an interface programmed for the older FLUOstar Galaxy software you just need to tell the interface to use the program Omega.exe from the Omega main directory instead of FLUO32.exe from the FLUOstar Galaxy main directory. If the DDE interface only accept a program file named FLUO32.exe you can rename the Omega file Omega.exe into FLUO32.exe. (Disadvantage: usage of the old and the new software together of this computer will no longer be possible.)

If the existing DDE interface uses the registry to automatically find the DDE server program you should create a registry key pointing to the new software:

'`HKEY_LOCAL_MACHINE\SOFTWARE\BMG Labtechnologies\FLUOstar\<main version number>\FLUOApp\MainDir`'

If FLUOstar Galaxy software is installed on your computer use a *`<main version number>`* which is higher than the existing (e.g. 5), otherwise use 4.

Enter as value of this key the path to the new software, usually '`C:\Program Files\BMG\Omega`'.

Set the value of the '**AsDDEServer**' parameter in the Omega configuration file 'Omega.ini' (section [ControlApp]) to 'true'.

BMG LABTECH GmbH
Allmendgrün 8
D-77799 Ortenberg
Germany
Tel. +49 / 781 / 96 96 8 – 0
Fax +49 / 781 / 96 96 8 – 67
e-mail germany@bmglabtech.com

Copyright © 2007-2014 BMG LABTECH. All rights reserved. All BMG LABTECH brand and product names are trademarks of BMG LABTECH. Other brand and product names are trademarks or registered trademarks of their respective holders.