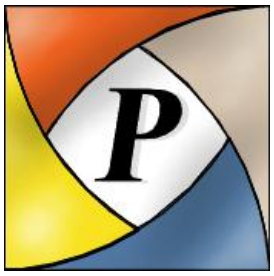
Institute for Energy Technology
OECD Halden Reactor Project



ProcSee

Graphical User Interface Management System

OPC Plugin

User's Guide
Reference Manual

1.3

This document will be subjected to revisions in the future as the development of ProcSee continues. New versions will be issued at new releases of the ProcSee system.

Published by : Institute for Energy Technology, OECD Halden Reactor Project
Date : June 2010
Revision : 1.3

Contents

Introduction

About OPC

OPC has become a de facto standard in industrial automation. Its development is managed by the OPC Foundation (www.opcfoundation.org). OPC Foundation states that OPC is "open connectivity via open standards".

OPC is a series of standards specifications. The first, and by far most widely used, is OPC Data Access which specifies the protocol to be used between OPC-compliant clients and servers in order for the client to read and write data item values.

About this manual

This manual describes ProcSee's OPC plugin and OPC configurator. The OPC plugin enables ProcSee's Run-Time Manager to act as an OPC Data Access client, reading and writing data item values directly from/to an OPC Data Access server. The OPC configurator is the GUI designers' primary tool to configure the data transfer between ProcSee and OPC servers.

This manual is divided into two parts; a User's Guide and a Reference Manual. The User's Guide part describes how to use the OPC configurator, how to load and identify the OPC plugin, and how to configure your computer for using the OPC plugin. The Reference Manual part describes in detail the format of the OPC plugin configuration file and all pTALK functions provided by the OPC plugin.

Part I

User's Guide

2

OPC Plugin

ProcSee's OPC plugin enables the RTM to act as an OPC Data Access client. The plugin uses OPC Data Access 2.0, allowing the RTM to access any server compliant with the OPC Data Access 2.0 specification.

Item browsing is an optional feature for OPC Data Access servers. Naturally, the browse-functions of the OPC plugin will only work if the server supports item browsing.

The OPC plugin is available in the Microsoft Windows version only.

Features

The following features are supported by the OPC plugin:

- Browsing for OPC servers on a specified host
- Browse for items and item properties within a specified OPC server
- Read and write item values
- Read item property values
- Automatic reconnect in case of communication failure

The main task of the OPC plugin is to read data item values from one or more OPC servers and copy the values into ProcSee variables where it can be used for visualisation. To establish the connections between OPC items and ProcSee variables is referred to as configuring the OPC plugin. At design time, when building operator interfaces, the primary tool to establish the connections between OPC items and ProcSee variables is the OPC configurator described in chapter 3. At run-time, when operators use the interfaces to monitor and control their process, the connections are established by reading a configuration file (file extension .popc) produced when the designer saved his/her work.

The OPC plugin offers several strategies to receive updated values from the OPC server. Subscription is generally the preferred method, but polling may also be used. When data is received from the server, the values are copied into the connected ProcSee variables. When the copying is completed, a pTALK statement will be executed. By default, this statement is "{ ::update(); }", but it can be any legal pTALK statement.

OPC items are subscribed to in groups, meaning that items in the same group are updated simultaneously. Groups can be enabled and disabled individually, and only items of enabled groups are updated. This feature can be used to reduce network traffic by disabling groups of items for which no pictures are currently displayed.

The OPC plugin can receive values of OPC item properties. Properties are normally static and will not change during run-time, but property values can be re-read using pTALK functions.

The OPC plugin also offers functionality to write item values to OPC servers using pTALK functions. Note that some OPC items may be read only and that properties are always read only.

Prerequisites

In order to use the OPC plugin the following requirements must be fulfilled.

ProcSee's OPC plugin and OPC Foundation's software for using OPC must be installed on the computer. By default, this is handled by the ProcSee installation. To verify that the plugin is installed right-click on your application item in GED's tree view, and select "Plugins…" from the menu. A dialog listing all loaded and registered plugins appears. Verify that the OPC plugin is listed among the registered plugins as shown in Figure 1.

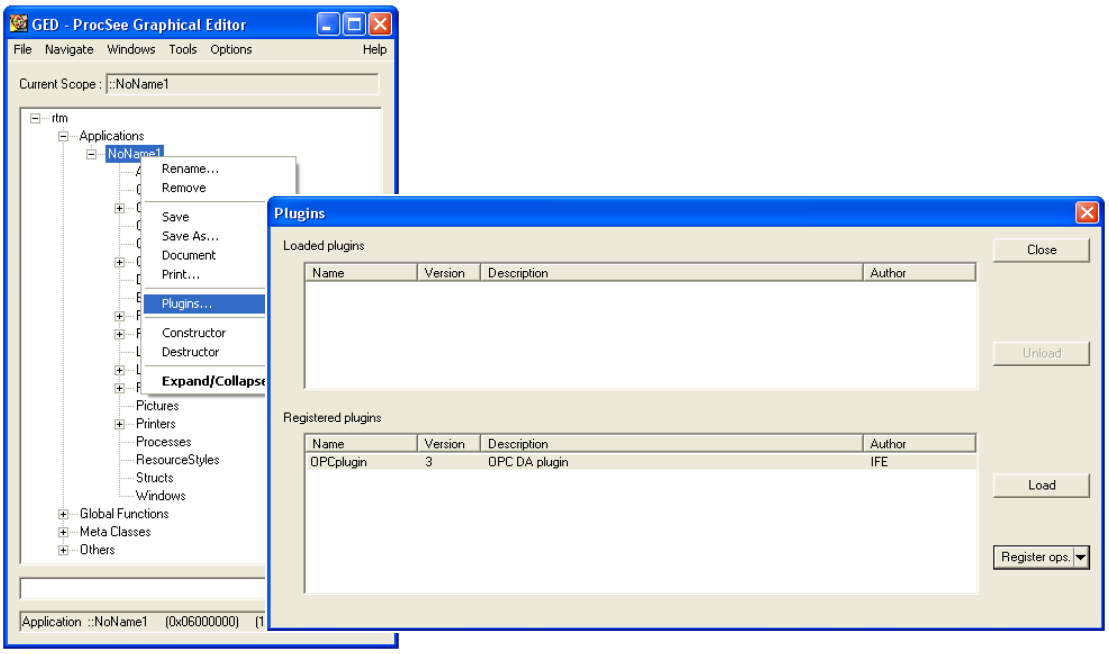
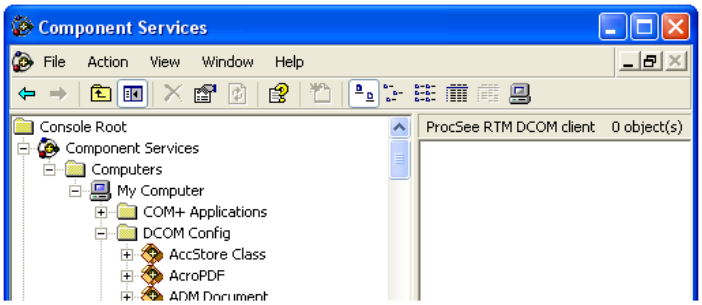


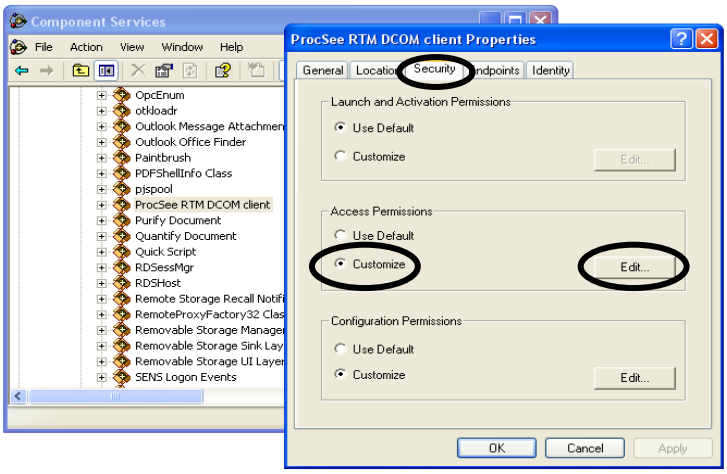
Figure 1 *Verifying installation of OPC plugin*

The client computer's default security settings are normally OK to receive updates and events from local OPC servers. However, to enable remote OPC servers to update values and issue events to the OPC plugin, the client computer's DCOM security settings must be altered. Altering the client computer's DCOM security settings is done using Microsoft's *dcomcnfg.exe* program. Its user interface differs between Windows XP and Windows 2000, Figure 2 describes the steps to go through for Windows XP and Figure 3 describes the steps for Windows 2000. The *dcomcnfg.exe* is normally located in *%SystemRoot%\system32*.

WinXP Step 1:
Start dcomcnfg.exe and expand the tree view as shown in the figure.

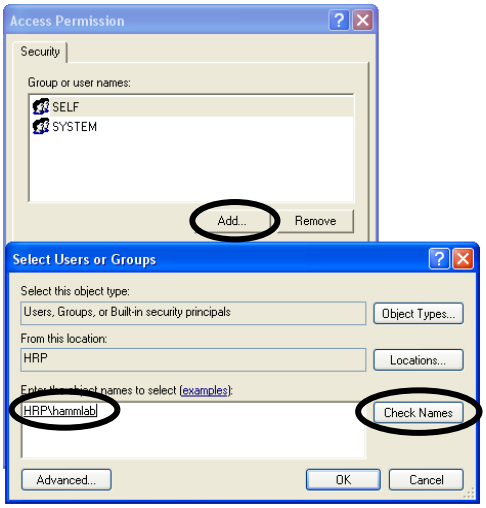


WinXP Step 2:
Scroll down the list to find "ProcSee RTM DCOM client". Right-click and select "Properties" from the popup-menu. Select the "Security" tab from the appearing dialog. In the "Access Permissions" field, select "Customize" and press the "Edit" button.



WinXP Step 3:
Press "Add" in the appearing dialog. Type the user-name of the process running the OPC server and press "Check Names". If the name is accepted, press "OK".

Typing the name of a group such as "Everyone" instead of a user-name is also possible



WinXP Step 4:
Select the new item and make sure that Remote Access is enabled. Press "OK".

Press "OK" in the dialog from Step 2.

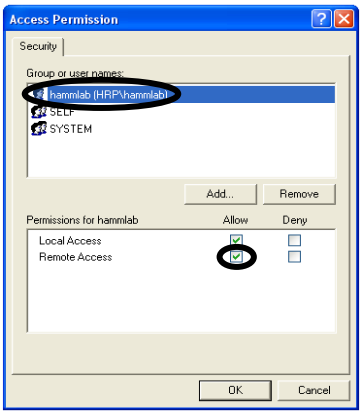
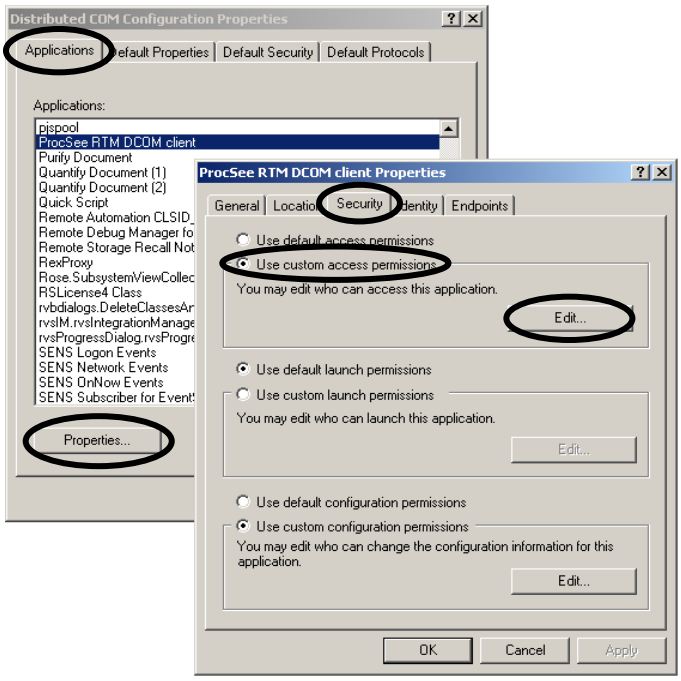


Figure 2 *Windows XP: Customised DCOM security settings*

Win2000 Step 1: Select the "Applications" tab and scroll down the list to find "ProcSee RTM DCOM client". Press the "Properties" button. In the appearing dialog, select the "Security" tab. Enable "Use custom access permissions" and press the "Edit…" button.



Win2000 Step 2: In the new dialog, press the "Add…" button. In the new dialog, press the "Show Users" button and scroll down the list to find the relevant user (optionally select "Everyone"). Press the "Add" button. Make sure that "Type of Access" is "Allow Access". Press OK.

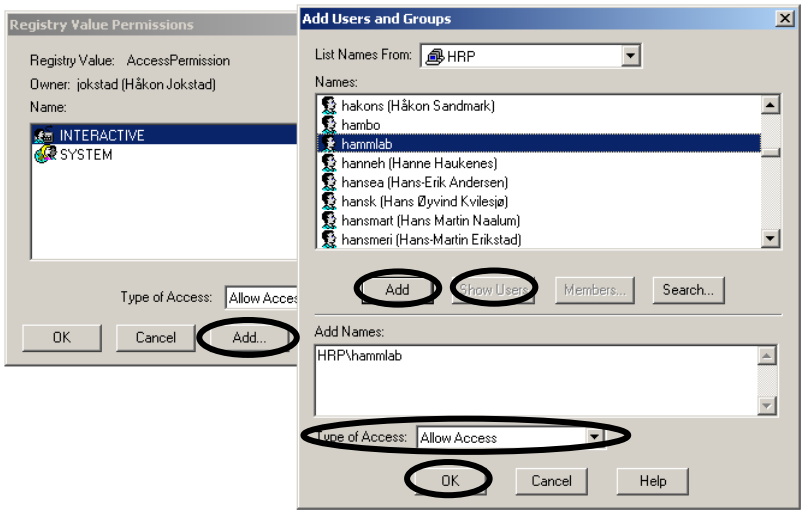


Figure 3 *Windows 2000: Customised DCOM security settings*

Loading the OPC plugin

Normally the OPC plugin is loaded into a ProcSee application using the OPC configurator as described in chapter 3. The OPC configurator adds the required pTALK code to the application, so when the application is later loaded, the OPC plugin is loaded as well. This section identifies what goes on behind the scene, and is provided here only as information for advanced users.

First, the OPC plugin is identified and loaded. In the application's Tdoc file, you will find the following entry:

```
application OPCtest
{
    …
    plugin "OPCplugin";
    …
```

The entry specifies that a plugin named OPCplugin should be loaded. ProcSee plugins are installed in the directory *%PROCSEE_DIR%\plugins\%ARCH%* and have file extension .pplg. When the plugin is loaded, it appears as a process called “OPC” in GED’s tree view.

Second, the plugin is provided with a licence and the validity of the licence is verified using the pTALK functions *initLicence* and *hasLicence*. A valid licence for the general OPC plugin is part of the ProcSee installation. It is very important that the licence registration function is called prior to any other OPC plugin function. The OPC plugin functions will fail if the licence registration function has not been called or an invalid licence is provided.

Third, the OPC plugin’s configuration file is loaded using the pTALK function *loadConfig*. The configuration file is the result of the designer’s work with the OPC configurator and is a text file with extension .popc. Refer to chapter 4 for a description of the format of the configuration file.

Reading Values From OPC Servers

The easiest way to receive item values from an OPC server is to use the OPC configurator described in chapter 3. Using the configurator, it is easy to browse for servers and items and to connect item values to ProcSee variables. Immediately after such a connection has been established, the ProcSee variable is regularly updated with values from the OPC server. Saving the work in the configurator ensures that whenever the ProcSee application is later reloaded, the ProcSee variables are regularly updated with values from the OPC server.

Enabling/Disabling Groups

Value updates from the OPC server can be enabled/disabled pr group. By default, value updates are enabled, but can be toggled using the pTALK function *enableGroup*. Disabling groups of items which are currently not displayed in any picture helps reducing network traffic and general system load.

```
// pTALK Example. Refer to chapter 5 for details
OPC.enableGroup( "Server1", "Group1", 0 );  // disable Group1
OPC.enableGroup( "Server1", "Group2", 1 );  // enable Group2
```

Writing Values to OPC Servers

To write an item value to an OPC server, the pTALK function *writeItemValue* must be used. In order to use this function, the item must be part of the OPC plugin's configuration, i.e. must be included in the OPC plugin's configuration file (.popc).

The *writeItemValue* function’s parameters are the new values and the identification of the item. The item's identification is a triple of names; 1) server name, 2) group name, 3) item name. Note that when creating item connections using the OPC configuration tool, the items by default are anonymous. See section *Renaming Items* on page 21, for information on how to name items using the configuration tool.

```
// pTALK Example. Refer to chapter 5 for details
OPC.writeItemValue( "Server1", "Group1", "Item1", 123 );
```

Updating Properties

Properties are meant to be static values, so clients need just to retrieve their values once. The values are transferred initially when connected in the configurator or when loading the OPC plugin's configuration file. However, property values can be changed on the server, and the OPC plugin offers a function to update property values, *updateItemPropertyValue*.

```
// pTALK Example. Refer to chapter 5 for details
int propId = 1234;  // The property's prop-attribute in .popc file
OPC.updateItemPropertyValue("Server1", "Group1", "Item1", propId, 4);
```

Troubleshooting

This section identifies potential problems when accessing OPC servers using the ProcSee OPC plugin and provides solutions.

Basics

Accessing local OPC servers are less subject to problems than accessing remote servers. When you have problems accessing a remote OPC server, you can if possible test the client at the OPC server computer, to check that the OPC server is working. Also verify that the user account used for the client has permission to access the OPC server.

For remote data communication, OPC is based on Microsoft's DCOM technology. There are several pitfalls involved concerning DCOM and DCOM security settings. A general advice if you experience problems to access an OPC server, is first to try to access the OPC server using a test-client from the server vendor. Make sure this test is carried out using the same client computer and the same user account as intended for ProcSee.

If accessing a remote OPC server, remember to alter the client computer's DCOM security settings as described in section *Prerequisites* on page 11.

Potential Problem 1: Fail to start OPC configurator

The OPC configurator requires Microsoft .NET Framework 1.1 + ServicePack 1. We recommend to use Microsoft .NET Framework 2.0.

The table below describes possible messages provided if the OPC configurator fails to start:

Message	Description
The OPC plugin is probably not installed.	You have not installed the OPC plugin, or the file %PROCSEE_DIR%\etc\winArch___ProcSeeOPCConfigTool.plib is missing. Re-install ProcSee. If you use custom installation, verify that the OPC checkmark is checked.
The OPC configuration tool is not properly registered in the registry.	Try to register the OPC configurator in the registry. From the directory %PROCSEE_DIR%\bin\winArch\components\1.0, run: regasm.exe /tlb OPCConfigTool.dll. regasm.exe is usually located in %SystemRoot%\Microsoft.NET\Framework\v1.1.4322
The OPC configuration tool is not properly registered in the Global Assembly Cache.	Try to register the OPC configurator in the Global Assembly Cache. From the directory %PROCSEE_DIR%\bin\winArch\components\1.0, run: gacutil.exe /i OPCConfigTool.dll gacutil.exe is usually located in %SystemRoot%\Microsoft.NET\Framework\v1.1.4322
Please check your ProcSee version for OPC support.	The OPC configurator is working, but the OPC plugin is not loaded. The OPC plugin is missing, or not registered. The OPC plugin should be located in: %PROCSEE_DIR%\plugins\winArch\OPCplugin.pplg How to register the plugin: Right click on your application in GED's main tree and select "Plugins…" In the appearing dialog, if the OPC plugin is listed among the registered plugins, press the button "Register ops" and then "Update registers". If not listed, press the button "Register ops." and then "Register new plugin…"

Potential Problem 2: Fail to locate OPC servers at specified host

First, check the spelling of the host name.

Check that the OPC Server Enumerator program, *OpcEnum.exe*, is installed on the specified host. It is normally found in the %SystemRoot%\system32 directory. This program, from the OPC Foundation, is normally installed as part of the OPC server installation. It is also part of the ProcSee installation, so if the program is not found on the specified host, you can install ProcSee on this host to install *OpcEnum.exe*.

If *OpcEnum.exe* is installed but the configurator still fails to locate OPC servers, there is probably something blocking the DCOM communication. See the next section for some hints.

Potential Problem 3: Fail to connect to OPC server

Check that DCOM is enabled on the computers, by running dcomcnfg.exe, and checking the Default Properties. The check box for "Enable Distributed COM on this computer" must be checked, on both the server and client computers.

On Windows XP, Service Pack 2, the *dcomcnfg.exe* settings for COM Security (found on My Computer properties) contains some buttons labelled "Edit Limits..." Verify that the limits include "Remote Access" for the users involved, on both server and client computer.

Other hints:

- Check that the server’s launch and access permissions include the client’s user account
- When changing DCOM settings for the server, the server must be restarted
- When changing the security settings for ProcSee RTM DCOM client, the RTM must be restarted
- When changing any default security settings on a computer, the computer should be restarted
- Check that server and client computer is registered in the **same** Windows Domain.
- Firewalls, including the Windows Firewall, may block DCOM communication and must be opened up or disabled.

Potential Problem 4: Fail to browse for items

Check your OPC server documentation and verify that the server supports browsing. Further, make sure your OPC server supports the type of browsing you use in the configurator. If you use "Tree view", hierarchical browsing is used. If you use "List view", flat browsing is used. Note also that flat browsing may take very long time if the number of items at the OPC server is large.

Potential Problem 5: Fail to get variable updates

Check what user account the OPC server uses. Verify the client computer's DCOM security settings as described in section Prerequisites on page 11. Try to add "Everyone" to the list of users and groups and remember to allow for "Remote Access".

3

OPC Configurator

The OPC configurator is the users' primary tool to configure the data transfer between the RTM and OPC servers. Using the OPC configurator, users can browse for OPC server, browse for items within the OPC servers, connect OPC items to ProcSee variables, and perform other tasks.

The OPC configurator is not a part of the OPC plugin, it is a separate software module using the features of the OPC plugin to create a configuration. It is a front-end to the OPC plugin at design time, in the same way as GED is a front-end to the RTM.

The result of working with the OPC configurator is a configuration file (.popc) as described in chapter 4. The configuration specified in such a file is later loaded each time the RTM loads the application.

Preparations

The purpose of using the OPC configurator is to connect OPC items to ProcSee variables, so make sure that at least one process with some variables is defined in your ProcSee application. Using the OPC configurator, OPC items can only be connected to ProcSee variables, not to attributes in an application or picture. (The OPC plugin, however, supports also connections to attributes.)

Of course, you also need access to an OPC server compliant with the requirements listed in chapter 2. The server must support item browsing. Several vendors provide of OPC servers free of charge for testing. See for instance www.matrikon.com.

If your OPC server runs on a remote host, make sure you open for the server to initiate updates and invoke functions in the client by altering the DCOM security settings. Refer to chapter 2 for details.

Starting the Configurator

To start the configurator, select an application in GED's tree view, then open the "Tools" menu and select "OPC Configuration".

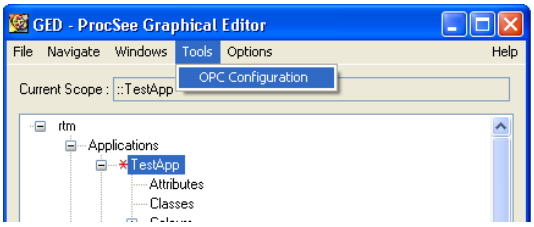


Figure 4 *Starting the configurator*

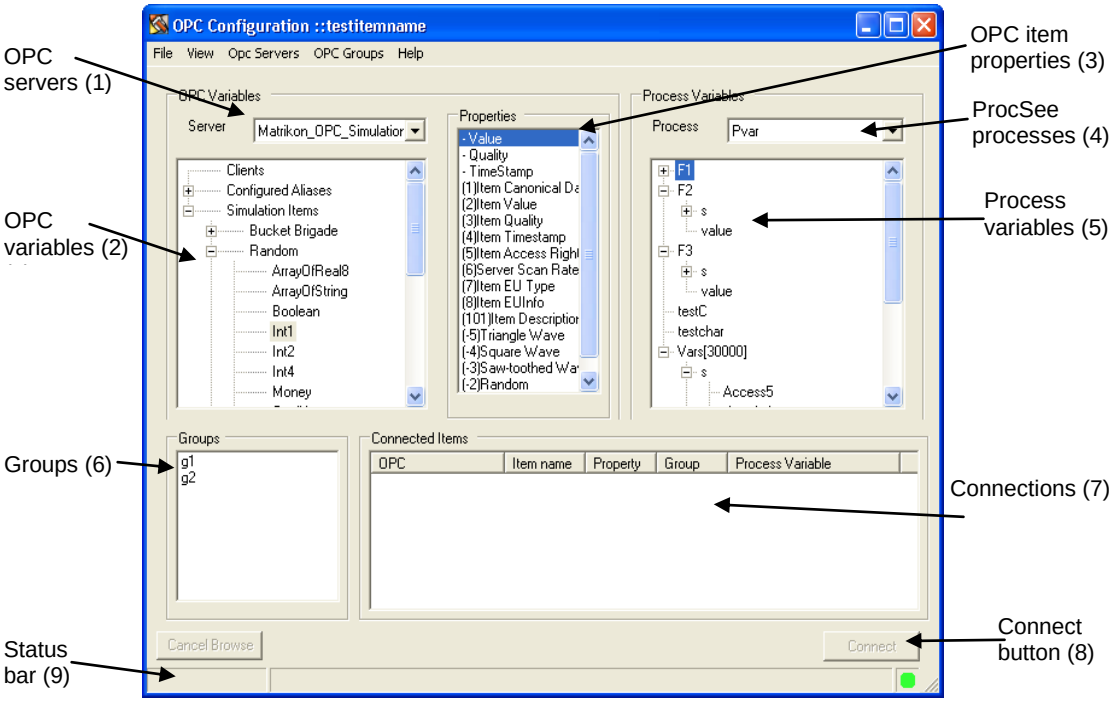


Figure 5 *The configurator's main dialog*

The configurator's main dialog is shown in Figure 5. It contains:

- A menu bar
- A combo-box listing the available OPC servers (1)
- A list of OPC items within a specific server (2)
- A list of OPC properties for the selected OPC item (3)
- A combo-box listing the available ProcSee processes (4)
- A list of ProcSee variables within the specified process (5)
- A list of groups for the selected OPC server (6)
- A list of connections for the selected OPC item (7)
- A connect button (8)
- A status bar (9)

Initially the lists will be empty. To start configuring, the first steps should be to add one or more servers and to add one or more groups per server as explained in the next sections.

Adding an OPC Server

In the menu bar, select "OPC Servers", and then "Add Server". The following dialog appears.

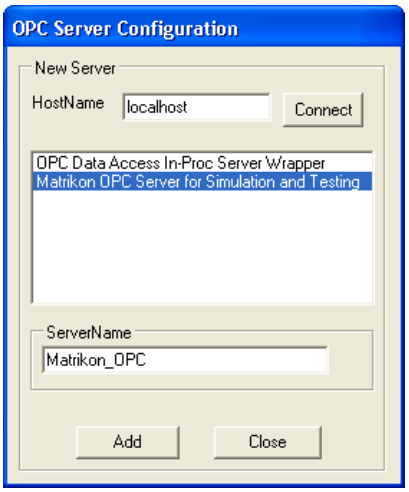


Figure 6 *Add OPC server dialog*

Type a hostname and press the "Connect" button to browse for servers at the specified host. A blank hostname, or *localhost*, will browse the local computer. After a few seconds, available OPC servers will be presented in the list.

Click on the server you want to connect to. The "ServerName" field at the bottom is filled with a proposed name to use for the server. Feel free to change the name according to your preferences, but note that the name must be a valid ProcSee identifier. Press the "Add" button to add the server to your configuration.

In the configurator's main dialog, the new server should now be included in the combo-box listing all available servers.

Adding a Group

Groups are defined per server, so in order to add a group an OPC server must be selected in the combo-box in the configurator's main dialog. To add a group select "OPC Groups" from the menu bar and then "Add Group". The following dialog appears.

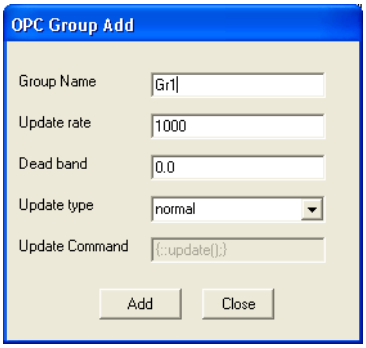


Figure 7 *Add OPC group dialog*

Type a name for the group and specify values for the other attributes in the dialog. The attributes are described in detail in chapter 4. Press the Add button to add the group.

The name of the group can not be changed later using the configuration tool, but attribute values can be changed using "Edit Group" from the "OPC Groups" menu.

Connecting an OPC Item to a ProcSee Variable

To connect an OPC item to a ProcSee variable the following steps should be taken:

1. In the combo-box at the upper left, select the desired OPC server

2. In the upper left list, select the desired OPC item
3. In the upper middle list, select the desired property for the selected item
4. In the upper right combo-box, select the process containing the desired ProcSee variable
5. In the upper right list, select the desired ProcSee variable
6. In the lower left list, select the desired group
7. Press the "Connect" button to connect the selected property of the selected OPC item to the selected ProcSee variable

When the "Connect" button is pressed, the new connection is added to the lower right list, and the selected ProcSee variable will be regularly updated with the value of the selected OPC item.

Browsing

From the view menu, the user can toggle between tree view and list view for OPC items and ProcSee variables. List view should be used with care if the set of items or variables is large as it may take a long time to obtain the entire list. Note also that the OPC server may not support hierarchical browsing required by the tree view.

The lists of OPC items and ProcSee variables can be refreshed using "Refresh OPC Variables" or "Refresh Process Variables" in the view menu.

Status and Error Messages

The status bar is divided into 3 panels as shown in Figure 8.

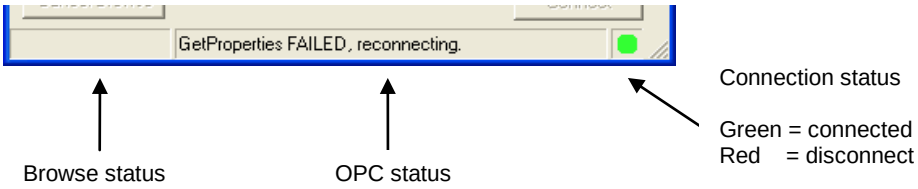


Figure 8 *status bar*

The left panel indicates "Browsing" when the configurator waits for the server to respond to a browse request. To cancel an ongoing browsing, press the "Cancel Browse" button.

The middle panel displays the last error or status message. The following messages may appear:

Message	Description
Get Properties failed	The server does not support properties on the selected OPC item.
GetProperties failed, reconnecting	The OPC plugin tried to get properties, but failed and lost connection to the server. The plugin tries to reconnect.
Server <serverName> disconnected	Lost connection to server <serverName>
Disconnected from server	Connection to the selected OPC server lost
Connected to server	Connection to the selected OPC server OK

The right panel displays an icon showing the connection status for the selected OPC server.

Deleting a Connected Item

To delete a connected item, first select an OPC item. Its connections will be displayed in "Connected Items". Select one connected item from the list, right click, and select "Delete".

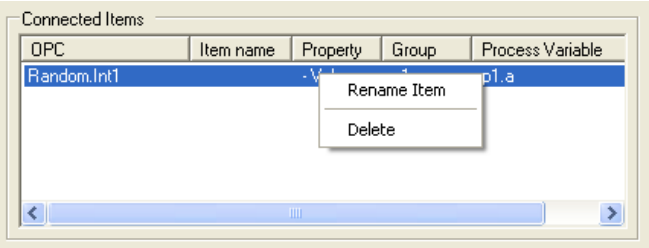


Figure 9 *Deleting a connected item*

When deleting an OPC server from a configuration, all connected items for this sever will be lost. When deleting a group, all connected items using this group will be deleted.

Renaming Items

All new items created by the OPC configurator are by default anonymous. To name or rename an item, right-click an item in the "Connected Items" list and select "Rename Item" as indicated in Figure 9. By giving the item a name, it is easier to write to the OPC item, using the function *writeItemValue*.

The OPC configurator can not be used to name or rename properties. Property names can optionally be entered directly in the .popc configuration file.

Editing Server Attributes

To edit the attributes of an OPC server, select "Edit Server" from the "OPC Servers" menu. The following dialog appears. The attributes are described in detail in chapter 4.

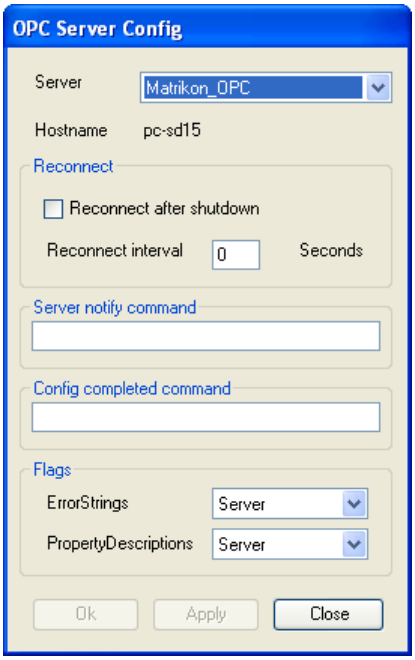


Figure 10 *OPC Server Config*

Saving the OPC Configuration

Select "Save" or "Save As" from the file menu to save your work. When selecting "Save As", the following dialog will appear.

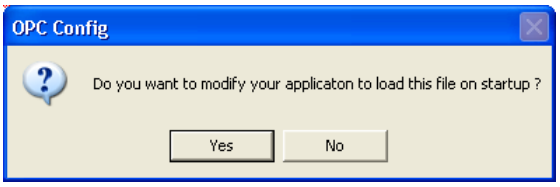


Figure 11 *Modify application question*

Answering “Yes” makes the configurator insert code in your application to automatically load the configuration every time the application is loaded. Remember to save the application from GED’s tree view after answering Yes. The modifications inserted into the application code are as follows:

- An attribute *OPCConfigFile* and a function *initOPCplugin* are added to the application
- The application’s constructor is modified to invoke the *initOPCplugin* function
- The application is modified to load the OPC plugin

Refer to section *Loading the OPC plugin* on page 13 also.

Part II

Reference Manual

OPC Plugin’s Configuration File

This chapter describes the format of the OPC plugin’s configuration file. By convention, the file extension is .popc.

The configuration file format resembles the .Tdoc format. Comments can start with '#' in addition to the familiar '//' and '/* */'. All OPC names, i.e. the value of all attributes called *id*, are specified as UTF-8 encoded strings.

Example

The file format is illustrated by the following example:

```
opcConfig
{
   server S1
   {
      id = "Matrikon.OPC.Simulation.1";
      hostname = "myhost";
      reconnectInterval = 60;                  // seconds
      reconnectAfterShutdown = 1;              // 1=true, 0=false
      connectNotify = "{MyOpcNotify(%S);}";    // default is "{::update();}"
      configCompleted = "{MyConfigCompleted(%S);}";  // default is "".
      flags = "errorstrings:client, propertydescriptions:both";

      group G1
      {
         updateType = "normal";  // subscription
         updateRate = 1000;      // milliseconds
         deadBand = 2.5;         // percent of engineering unit range
         updateCommand = "{MyOpcUpdated(%S,%G);}"; // default is "{::update();}"

         item I1
         {
            id = "Saw-toothed Waves.Real4";
            v = "MyProcess.Var1.value";      // Item value
            q = "MyProcess.Var1.quality";    // Item quality
            t = "MyProcess.Var1.theTime";    // Item time

            property unitProp
            {
                prop = 1234;
                v = "MyProcess.Var1.engUnit";
            }

         } // end item I1

      } // end group G1

      property serverProp
      {
         id = "Saw-toothed Waves.Real8";
         prop = 345;
         v = "MyProcess.MyVar";
      }

   } // end server S1

   server S2
   {
      ...
   }

} // end opcConfig
```

Syntax

The file starts with the keyword *opcConfig* to identify it as an OPC plugin configuration file.

Within the opcConfig scope, there may be zero or more servers. Each server is identified with the keyword *server* and an optional name.

Within the server scope, there is a set of server attributes followed by zero or more groups and zero or more properties. Each group is identified by the keyword *group* and a mandatory name. Each property is identified by the keyword *property* and an optional name.

Within the group scope there is a set of group attributes followed by zero or more items. Each item is identified by the keyword *item* and an optional name.

Within the item scope there is a set of item attributes followed by zero or more item properties. Each item property is identified with the keyword *property* and an optional name.

Within the property and item property scopes there is a set of attributes.

Attributes

The following attributes exist for a server specification:

Server attribute	Mandatory	Description
id	mandatory	The program identifier (ProgId) or class identifier (CLSID) of the OPC server
hostname	optional	Host where OPC server is running, name or IP address. Default value is an empty string, meaning local host
reconnectInterval	optional	Number of seconds between each time the plugin should attempt to re-establish a broken connection with the OPC server. Default value is 0, meaning that the plugin will not try to reconnect
reconnectAfterShutdown	optional	Boolean value, 0 = false, 1 = true. Specifies whether the plugin should reconnect to the OPC server if the server issues a shutdown message. If the value is true, the plugin will try to reconnect at intervals specified by *reconnectInterval*
connectNotify	optional	A pTALK statement to be executed whenever connection with the OPC server is established or lost. The statement is executed within the scope of the OPC process. Default value is "{::update();}". Any '%S' in this string will be replaced with the index of the server.
configCompleted	optional	A pTALK statement to be executed whenever the configuration has been completed. That is each time the applyOPCChanges function has done its work. The statement is executed within the scope of the OPC process. If not specified, nothing is done to signal that the configuration is completed. Any '%S' in this string will be replaced with the index of the server.

flags	optional	Additional flags, like:
		`errorstrings: server,client,both`
		`propertydescriptions: server,client,both`
		Default: `"errorstrings=server, propertydescriptions=server"`.
		The value `both` means to first try the client, and if the client is not able to find a string, then try the server.

The following attributes exist for a group specification:

Group attribute	Mandatory	Description
updateType	optional	Legal values are "normal", "poll", "async" and "sync". Default value is "normal". If value is "normal", a subscription strategy is used and the item values are updated at an interval specified by *updateRate*. The

		server may reject the update interval and specify a slower rate, the actual update rate can be determined with the pTALK function *getGroupRealUpdateRate*. With the subscription strategy, only values that has changed since the last update is transferred, and the *deadBand* attribute can be set to avoid insignificant updates. If value is "poll", the plugin polls the server for new values at an interval specified by *updateRate*. Minimum value for *updateRate* is 250 ms, and the value is forced to be a multiple of 250. The actual value can be determined with the pTALK function *getGroupRealUpdateRate*. Polling generally requires more resources than subscription, because all values are transferred each update. If value is "async" or "sync", the items in the group will receive no updates from the server. Useful for groups of items where the plugin should only write item values, not read. If value is "async", asynchronous writing is used. If value is "sync", synchronous writing is used.
updateRate	mandatory if *updateType* is "normal" or "poll"	Number of milliseconds between each data update to be received from the OPC server. The attribute is used only if *updateType* is "normal" or "poll".
deadBand	optional	The attribute is used only if *updateType* is "normal", and can be used to ignore insignificant value changes. The value denotes a percentage of the range between the items' highEU and lowEU properties. Default value is 0, meaning all changes are significant.
updateCommand	optional	A pTALK statement to be executed after each variable update. The statement is executed within the scope of the OPC process. Default value is "{::update();}". The attribute is used only if *updateType* is "normal" or "poll". Any '%S' in this string will be replaced with the index of the server. Any '%G' in this string will be replaced with the index of the group.

The following attributes exist for an item specification:

Item attribute	Mandatory	Description
id	mandatory	Name of item at the OPC server
v	optional	Name of the ProcSee variable or attribute to be updated by the item's value. The data type of the variable or attribute can be float, double, int, short int, or char*. The variable can also be a one dimensional array of one of these types, or a pointer to one of these types (array or pointer requires the OPC item to be an array). If the ProcSee variable is a pointer, the pointer is set to point to a new array each time the value is received. The number of elements in the array is given by the pTalk function arrayLength. If the ProcSee variable is an array, the array elements received from the OPC server is put into this array. If the number of elements received is less than the number of elements of the ProcSee array variable, the rest of the elements are unchanged. If the number of elements received is larger than the number of elements of the array, the excess elements are ignored.
vt	optional	Type to request from the OPC server, overriding the type of the variable specified in the v attribute. This attribute is mainly for testing the value conversions in the OPC plugin, and should not be used in normal conditions. The possible values of this attribute are: "I1", "I2", "I4" (int), "UI1", "UI2", "UI4" (unsigned int), "R4" (float), "R8" (double), "BSTR" (string), or

		"ARRAY|x" where x is one of the previous values, e.g.: "ARRAY|R4". When this attribute is not specified, the type requested from the OPC server is the type of the ProcSee variable specified in the v attribute.
q	optional	Name of the ProcSee variable or attribute to be updated by the item's quality. The quality value is a 16 bit integer. The upper 8 bits are for vendor specific use. The 8 low bits are arranged as follows: QQSSSSLL, where QQ is quality, SSSS is sub status, and LL is limit status. The QQ bits have the bit value 00SSSSLL meaning Bad, 01SSSSLL for Uncertain, and 11SSSSLL for Good. The 10SSSSLL values are not used by OPC. For more information about the quality bits, refer to the OPC Data Access Custom Interface Specification (2.05A or 3.00) chapter 6.8.
t	optional	Name of the ProcSee variable or attribute to be updated by the item's time. The time is converted to an integer holding the number of seconds since ~1970 (unix time).

The following attributes exist for an item property specification:

Item property attribute	Mandatory	Description
prop	mandatory	Numeric id of the item's property at the OPC server
v	mandatory	Name of the ProcSee variable or attribute to obtain the item property's value. The value is obtained once when connection to the server is established, and may be updated later by calling pTALK function *updateItemPropertyValue*. The type of the variable should match the type expected for the given property number.

The following attributes exist for a property specification:

Property attribute	Mandatory	Description
id	mandatory	Name of item at the OPC server
prop	mandatory	Numeric id of the item's property at the OPC server
v	mandatory	Name of the ProcSee variable or attribute to obtain the property's value. The value is obtained once connection to the server is established, and may be updated later by calling pTALK function *updatePropertyValue*. The type of the variable should match the type expected for the given property number.

5

OPC Plugin Functions

This chapter describes all functions provided by the OPC plugin. These functions are exposed to the ProcSee RTM as pTALK functions.

General parameter descriptions

Several functions in the OPC plugin use a set of parameters to identify a server, a group, an item, or an item property. One parameter, serverName, is needed to identify a server. Two parameters, serverName and groupName, are needed to identify a group. Three parameters, serverName, groupName and itemName, are needed to identify an item. Four parameters, serverName, groupName, itemName and propertyName, are needed to identify an item's property. In general, any of these four parameters can be either a name (char*), an OPC identifier (char*), or an index (int). In the documentation, their data types are denoted **[char* | int]**.

Identification using names

When identifying servers, groups, items and properties by name, it is the configuration element names that are used by default. Examples of such names are `S1`, `G1`, `I1`, `unitProp`, and `serverProp`, found in the configuration file example on page 25. Two examples of how to use configuration element names are given below.

```
OPC.setGroupUpdateRate( "S1", "G1", 500 );

int p = OPC.getItemPropertyOpcProp( "S1", "G1", "I1", "unitProp" );
```

Identification using OPC identifiers

Servers, items and properties can also be identified using OPC identifiers. To support the various options for identification, the functions have an optional last parameter, *flags*, that is used to specify how the *serverName*, *itemName* and *propertyName* parameters should be interpreted. The optional parameter is a bit-mask where each bit controls how a specific parameter should be interpreted. The bit 0x1 controls parameter *serverName*, bit 0x2 controls *itemName*, and bit 0x4 controls *propertyName*. When the 0x1 bit is set, the *serverName* parameter should be the OPC server's ProgId or CLSID (server's *id* attribute in .popc file). When the 0x2 bit is set, the *itemName* parameter should be the OPC item id (item's *id* attribute in .popc file). When the 0x4 bit is set, the *propertyName* parameter should be the integer OPC property (property's *prop* attribute in .popc file). When a bit is unset, the configuration element name or index is used. The examples below demonstrate how to use the optional *flags* argument. The examples refer to the configuration specified in the example configuration file on page 25.

In cases where there is more than one configuration element with the same OPC identifier, the first one found will be used.

```
OPC.enableGroup( "S1",
                 "G1",
                 1 );  // Configuration element names
OPC.enableGroup( "Matrikon.OPC.Simulation.1",
                 "G1",
                 1,
                 0x1 ); // OPC ProgId identifier for server

OPC.writeItemValue( "S1",
                    "G1",
                    "I1",
                    3.14 ); // Configuration element names
OPC.writeItemValue( "Matrikon.OPC.Simulation.1",
                    "G1",
                    "Saw-toothed Waves.Real4",
                    3.14,
                    0x1|0x2 ); // OPC identifiers for server & item

OPC.updateItemPropertyValue( "S1",
                             "G1",
                             "I1",
                             "unitProp" ); // Configuration element names
OPC.updateItemPropertyValue( "Matrikon.OPC.Simulation.1",
                             "G1",
                             "Saw-toothed Waves.Real4",
                             1234,
                             0x1|0x2|0x4 ); // OPC identifiers
```

Identification using indexes

Servers, groups, items and properties can also be identified by indexes established by the ProcSee OPC plugin. An index is an integer in the range 1 to maxIndex, where maxIndex can be obtained using the functions *getMaxServerIndex*, *getMaxGroupIndex*, *getMaxItemIndex*, *getMaxItemPropertyIndex* or *getMaxPropertyIndex*. Note that indexes do not change at run-time, so an index may reference a removed server, group, item or property. All functions return a defined error value if the index (or name) refers to a non-existing server, group, item or property.

Argument errors

If the server, group, item, or property specified in the arguments are not found, an error message about this is given by all the functions, except the find...Ix functions. To check for existence of e.g. a server from code, the findServerIx function should be used, because this only returns -1 if the server is not found, whereas all other functions that looks up a server, returns an error value and gives an error-message when the server is not found.

Example of using indexes to access configuration items:

```
int serverMax = getMaxServerIndex();
for ( int s = 1; s <= serverMax; s++ )
{
     if ( findServerIx( s ) < 1 )
          continue; // Non-existing server. Continue to next

     int groupMax = getMaxGroupIndex( s );
     for ( int g = 1; g <= groupMax; g++ )
     {
          if ( findGroupIx( s, g ) < 1 )
               continue; // Non-existing group. Continue to next

          int itemMax = getMaxItemIndex( s, g );
          for ( int i = 1; i <= itemMax; i++ )
          {
               if ( findItemIx( s, g, i ) < 1 )
                    continue; // Non-existing item. Continue to next

               int propMax = getMaxItemPropertyIndex( s, g, i );
               for ( int p = 1; p <= propMax; p++ )
               {
                    if ( findItemPropertyIx( s, g, i, p ) < 1 )
                         continue; // Non-existing. Continue to next
                    // do something…
               }
          }
     }
}
```

Licence functions

The following functions are available for handling licencing issues for the OPC plugin. The *initLicence* function must be called prior to any other function in the plugin.

Function	Description
hasLicence	Get the licence status
initLicence	Initialize the OPC plugin with a license

F u n c t i o n s :

int hasLicence()

This function returns the licence status.

Example

```
if ( !hasLicence() )
{
   printf ( "Fatal error: No licence" );
}
```

Return Value

Returns 0 if no licence, non-zero if licence OK

int initLicence(char* licenceName, char* licenceData=0)

Initializes the OPC plugin with a licence.

Parameters

licenceName – Name of the licence. Must match the name of a .popl file in the %PROCSEE_DIR%/plugins/winArch directory

licenceData – Optional licence key data

Return Value

Returns 0 on failure, non-zero on success.

Example

```
initLicence( "OPCLic" );
```

Configuration and query functions

Functions described in this section are used to configure the OPC plugin, or browse the OPC server for items and properties.

Configuration functions

The following functions are for loading, saving and applying changes to the configuration.

Function	Description
applyOPCChanges	Updates the OPC servers with changes performed by other functions in the OPC plugin.
loadConfig	Loads configuration file.
saveConfig	Saves current configuration.

F u n c t i o n s :

int applyOPCChanges()

This function must be called to actually update the OPC servers with changes performed by the functions listed in the table below. It will among other things try to establish the connection to OPC servers that are not connected.

Functions
addGroup
addItem
addItemProperty
addProperty
addServer
setGroupDeadBand
setGroupUpdateRate
setGroupUpdateType

Return Value

Returns 0 on failure, non-zero on success.

int loadConfig(char* fileName)

Removes any previously registered configuration, and loads a configuration file of type .popc. Calls *applyOPCChanges* when the configuration is read.

Parameters

fileName – File name identifying a .popc file. Full path or relative to the .pctx file.

Return Value

Returns the number of errors found in the file, 0 if OK.

Example

```
int numErrors = loadConfig( "MyConfig.popc" );
```

int saveConfig(char* fileName)

Saves the current configuration to file.

Parameters

fileName – File name identifying a .popc file. Full path or relative to the .pctx file.

Return Value

Returns 0 on failure, and a non-zero value on success.

Example

```
int status = saveConfig( "MyConfig.popc" );
```

Browse functions

The following functions are used to browse for OPC servers and for items and properties within a specified server.

Function	Description
getItemIdFromBrowseName	Gets the full OPC item id for a specified browse item short name.
getItems	Browses for OPC items within a specified server.
getProperties	Browses for properties for a specified OPC item.
getServers	Browses for OPC servers on a specified host.

F u n c t i o n s :

int getItemIdFromBrowseName(
[char* | int] serverName,
char* parentOpcItemId,
char* shortName,
void (*nameReady) (char* opcItemId, any userData, int status),
any userData,
int flags = 0)

This function gets the OPC item id from the short name returned with the %D format in the *getItems* function. Avoiding the %I format for the *getItems* function will speed up browsing significantly.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

parentOpcItemId – The OPC item id of the item's parent in the hierarchy.

shortName – The short name returned by the %D conversion from *getItems*.

nameReady – The callback function to be invoked when the item's OPC item id is available

userData – Anything. Provided as input argument to the callback function.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

Example

```
OPC.getItemIdFromBrowseName( "MyServer", "Saw-Toothed Waves",
                             "Int1", ItemIdCb, "Int1" );

void ItemIdCb( char* opcItemId, char* shortName, int status )
{
   if ( status == 0 )
      return; // Browsing failed

   printf( "OPC item id of %s is %s", shortName, opcItemId );
}
```

int getItems([char* | int] serverName,
char* options,
char* filter,
List* itemList,
char* format,
void (*listReady)(List* itemList, any userData, int status),
any userData,
int flags = 0)

Browses for OPC items at the server specified by *serverName.* The *options* argument controls how to browse: flat or hierarchical, optionally starting from a known position in the hierarchy. The *filter* argument is passed on to the OPC server where the server may use it to return only those items

matching the filter. When browsing completes, the callback function *listReady* is invoked with the resulting list of items formatted according to the *format* argument.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

options – Controls how to browse. To browse for items using a flat structure, use the keyword *flat*. To browse for items using a hierarchical structure, use the keywords *branch* and/or *leaf*. Use the keyword *startBrowse:* to start browsing from a known location in the hierarchy, and apply a previously retrieved opcItemId as starting point. Note that some servers only support flat browsing, in such cases flat browsing is used regardless of the leaf and branch keywords. Examples of legal values: "flat" "leaf,branch" "leaf,branch,startBrowse:anOpcItemId"

filter – Optionally used by the OPC server to filter items. Only items matching the filter will be returned. Refer to your OPC sever documentation for details.

itemList – A list object to be filled with items matching the query. If 0 is supplied, ProcSee will create a new list object.

format – A text string specifying what information to return for each item. %I will be converted to the item's opcItemId, %D will be converted to the item's short name, and %N will be converted to "B" for branch nodes, "L" for leaf nodes, and "F" if flat browsing is used. Note that using %I requires an extra function call to the server pr item, and may lead to slow response if a large number of items are involved. By using %D to obtain a short name, the OPC item id can later be obtained using the function *getItemIdFromBrowseName*.

listReady – The callback function to be invoked when browsing completes.

userData – Anything. Provided as input argument to the callback function.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

Example 1

```
// Flat browsing to obtain all items in the server at once.
// This may take a long time for servers with many items.
OPC.getItems("MyServer", "flat", 0, 0, "%I", FlatCb, "MyServer");

// The callback function for the flat browsing request above
void FlatCb( List* flatList, char* userData, int status )
{
   if ( status == 0 )
      return; // Browsing failed

   int n = flatList.length();
   printf( "%d items found in OPC server %s", n, userData );
   for ( int i = 0; i < n; i++ )
   {
      char* opcItemId = flatList.get(i);
      printf( "OPC item id of item %d is: %s", i+1, opcItemId );
   }
}
```

Example 2

```
// Define a function to browse hierarchicaly
// from a specified starting point
```

```
void BrowseForItems( char* serv, char* startPos )
{
   char *options = newCharArray( 32+strlen(startPos) );
   sprintf( options, "leaf,branch,startBrowse:\"%s\"", startPos);

   OPC.getItems(serv, options, 0, 0, "%N|%I", ItemsCb, startPos);
}

// The callback function for the BrowseForItems function
void ItemsCb( List* items, char* userData, int status )
{
   if ( status == 0 )
      return; // Browsing failed

   int n = items.length();
   printf( "%d items found starting from %s", n, userData );
   for ( int i = 0; i < n; i++ )
   {
      char* s = items.get(i);
      char* N = strfield( s, 0, '|' );
      char* I = strfield( s, 1, '|' );
      char* nodeType = "Unknown";
      if ( *N == 'B' ) nodeType = "Branch node";
      if ( *N == 'L' ) nodeType = "Leaf node  ";
      printf( "   %s: %s", nodeType, I );
   }
}

// Invoke BrowseForItems from the top of the hierarchy
BrowseForItems( "MyServer", "" );

// Invoke BrowseForItems from a previously returned opcItemId
BrowseForItems( "MyServer", "Saw-toothed Waves" );
```

---

int getProperties([char* | int] serverName,
char* opcItemId,
char* options,
List* propList,
char* format,
void (*listReady)(List* propList, any userData, int status),
any userData,
int flags = 0)

Browses a specified item for properties. Arguments *serverName* and *opcItemId* identify the item. The *options* argument can be used to suppress errors from the OPC call. When browsing completes, the callback function *listReady* is invoked with the resulting list of properties formatted according to the *format* argument.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

opcItemId – Identifies the item. Corresponds to the value returned when the %I format is used for the function *getItems*. Also corresponds to the value of the *opcItemId* argument to the callback function when calling *getItemIdFromBrowseName*.

options – Some OPC servers fail if a client browses for non-existing properties. To suppress such error messages, use the value *"silent"*. Otherwise, use an empty string, or combine it with the other options by separating them with '|'. When the property is an array and the format

contains %V, the separator used between the array elements can be set with the option " arrayElementSeparator=';' ". If not set, array elements are separated with a comma (',').

propList – A list object to be filled with properties matching the query. If 0 is supplied, ProcSee will create a new list object.

format – A text string specifying what information to return for each property. %P is converted to the property id (an integer), %#P is the same, but the value is written as a hexadecimal unsigned number, %D is converted to the description, %#D is the same, but with the description in double quotes, %T and %U is converted to the type (as text) (%U is the type of the returned value, %T the type returned by the QueryAvailableProperties OPC function), %#T and %#U is the same as %T and %U, but the type is given as a hexadecimal number with the VARTYPE type values used in the Windows VARIANT type, %I is converted to the OPC item id and %V is converted to the value (as text). If the value is an array, up to the first 18-20 values are inserted into the string.

listReady – The callback function to be invoked when browsing completes.

userData – anything. Provided as input argument to the callback function.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

Example

```
{
   char* anOpcItemId = "Saw-toothed Waves.Int4";
   OPC.getProperties( "MyServer", anOpcItemId,
                      "", 0, "%P|%D|%T|%V", PropCb, anOpcItemId);
}

void PropCb( List* props, char* userData, int status )
{
   if ( status == 0 )
      return; // Browsing failed

   int n = props.length();
   printf( "%d properties found for %s", n, userData );
   for ( i = 0; i < n; i++ )
   {
      char* s = props.get(i);
      char* P = strfield( s, 0, '|' );
      char* D = strfield( s, 1, '|' );
      char* T = strfield( s, 2, '|' );
      char* V = strfield( s, 3, '|' );
      printf( "   propId=%s, desc=%s, type=%s, val=%s",
              P, D, T, V );
   }
}
```

int getServers(char* hostName,
char* options,
List* serverList,
char* format,
void (*listReady)(List* serverList, any userData, int status),
any userData)

Browses for OPC servers on the host specified by *hostName* using the OPCenum.exe program. When browsing completes, the callback function *listReady* is invoked with the resulting list of servers.

Parameters

hostName – The host to browse for OPC servers. Host name or IP address can be specified.

options – Reserved for future use. An empty string should be supplied.

serverList – A list object to be filled with servers matching the query. If 0 is supplied, ProcSee will create a new list object.

format – A text string specifying what information to return for each server. The format can contain %C for CLSID, %P for ProgID and %U for UserType (a user-readable text for the server). If the format parameter is empty, the format "%C|%P|%U" will be used.

listReady – The callback function to be invoked when browsing completes.

userData – Anything. Provided as input argument to the callback function.

Return Value

Return 0 on failure, non-zero on success.

Example

```
{
   // Browse for servers. Using host name as userData
   int sts = OPC.getServers( "aHost", "", 0, "%C|%P|%U",
                             ServersCb, "aHost" );
}

// The callback function
void ServersCb( List* sl, char* host, int status )
{
   if ( status == 0 )
   {
      printf( "Browsing for servers on %s failed", host );
      return;
   }

   for ( int i = 0; i < sl.length(); i++ )
   {
      char* s = sl.get(i) ;
      char* C = strfield( s, 0, '|' );
      char* P = strfield( s, 1, '|' );
      char* U = strfield( s, 2, '|' );
      printf( "OPC server %d on %s: ", i+1, host );
      printf( "   CLSID    = %s", C );
      printf( "   ProgID   = %s", P );
      printf( "   UserType = %s", U );
   }
}
```

Server functions

The following functions are used to configure servers.

Function	Description
addServer	Adds a server to the current configuration.
getMaxServerIndex	Gets maximum server index.
getServerConfigCompletedCommand	Gets the configCompleted attribute of a server.
getServerFlags	Gets the flags attribute of a server.
getServerHostName	Gets the host name of a server.
getServerName	Gets the server name.
getServerNotifyCommand	Gets the connectNotify attribute of a server.
getServerProgId	Gets the ProgId of a server (The id attribute).
getServerReconnectAfterShutdown	Gets the reconnectAfterShutdown attribute of a server.
getServerReconnectInterval	Gets the reconnectInterval attribute of a server.
findServerIx	Finds server index from server name.
isServerConnected	Gets the connection status of a server.
removeServer	Remove a server from the current configuration.
renameServer	Renames a server.
setServerConfigCompletedCommand	Sets the configCompleted attribute for a server.
setServerFlags	Sets the flags attribute for a server.
setServerNotifyCommand	Updates the connectNotify attribute for a server.
setServerReconnectAfterShutdown	Updates the reconnectAfterShutdown attribute for a server.
setServerReconnectInterval	Updates the reconnectInterval attribute for a server.

F u n c t i o n s :

---

int addServer(char* serverName,
char* opcServerProgId,
char* hostName = 0)

Adds a new OPC server to the current configuration. The function *applyOPCChanges* must be called after adding a server, to effectuate the changes and connect to the server.

Parameters

serverName – The name to be used by ProcSee to identify the new OPC server.

opcServerProgId – The OPC server's ProgId (UTF8 encoded) or CLSID. For instance "Matrikon.OPC.Simulation.1" or "{604F216F-5893-4543-92B0-C1AE287C585C}".

hostName – Name of the host running the OPC server. If empty, localhost is assumed.

Return Value

Returns 0 on failure, non-zero on success.

Example

```
int status = addServer( "MyServer",
                        "{604F216F-5893-4543-92B0-C1AE287C585C}",
                        "aHostName" );
```

int getMaxServerIndex()

Returns the maximum server index. Removed servers may be included in the range 1..max.

Return Value

Returns the maximum server index , -1 on failure.

Example

```
  int maxS = OPC.getMaxServerIndex();
  for ( int s = 1; s <= maxS; s++ )
  {
       if ( OPC.findServerIx( s ) >= 1 )
       {
            char* serverName = OPC.getServerName( s );
            printf( "%s", serverName );
       }
}
```

char* getServerConfigCompletedCommand([char*|int] serverName, int flags=0)

Gets the configCompleted attribute from the server given by *serverName*. The config completed command is pTALK code executed when the configuration has been completed by the applyOPCChanges function for this server.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the configCompleted pTalk code. This is 0 if not set, or if serverName not found.

Example

```
char* pTalkCode = getServerConfigCompletedCommand( "MyServer" );
```

See Also

Section *Attributes* on page 26.

char* getServerFlags([char* | int] serverName, int flags = 0)

Returns the server's flags attribute.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the flags attribute, null pointer on failure.

See Also

Function *setServerFlags* on page 44 and section *Attributes* on page 26.

---

char* getServerHostName([char* | int] serverName, int flags = 0)

Returns the server's host name.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the host name, null pointer on failure.

---

char* getServerName([char*| int] serverName, int flags = 0)

Returns the server's ProcSee name.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29. Normally an index or ProgId/CLSID.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the server's ProcSee name, null pointer on failure.

---

char* getServerNotifyCommand([char* | int] serverName, int flags = 0)

Returns the value of the server's *connectNotify* attribute.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the pTALK code stored in attribute *connectNotify*.

See Also

Section *Attributes* on page 26

---

char* getServerProgId([char* | int] serverName, int flags = 0)

Returns the server's ProgId or CLSID. (The id attribute of the server configuration.)

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the server's ProgId or CLSID, null pointer on failure.

---

int getServerReconnectAfterShutdown([char* | int] serverName, int flags = 0)

Returns the value of the server's *reconnectAfterShutdown* attribute.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 1 if the OPC plugin will try to reconnect when a shutdown message is received from the server, otherwise 0.

See Also

Section *Attributes* on page 26

---

int getServerReconnectInterval([char* | int] serverName, int flags = 0)

Returns the value of the server's *reconnectInterval* attribute.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the number of seconds between each time the OPC plugin will try to reconnect to the server if the connection is broken. 0 if the plugin will not try to reconnect.

See Also

Section *Attributes* on page 26

---

int findServerIx([char* | int] serverName, int flags = 0)

Returns the server index. Gives no error messages if server not found.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the server index (>= 1) if the server is found, returns <= 0 if not found.

---

int isServerConnected([char* | int] serverName, int flags = 0)

Returns the connection state.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 1 if the OPC server is connected, otherwise 0.

---

int removeServer([char* | int] serverName, int flags = 0)

Removes the server from the current configuration.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

---

int renameServer([char*|int] serverName, char* newName, int flags = 0)

Sets a new ProcSee name for the server.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

newName – The new server name

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

Example

```
int status = renameServer( “MyServer”, ”NewServerName” );
```

int setServerConfigCompletedCommand([char*|int] serverName, char* configCompletedCommand, int flags=0)

Sets the configCompleted attribute in the server given by *serverName* to *configCompletedCommand*. The config completed command is pTALK code executed when the configuration has been completed by the applyOPCChanges function for this server.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

configCompletedCommand – The new value of the *configCompleted* attribute in the server.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

Example

```
int sts = setServerConfigCompletedCommand( "MyServer",
              "{OnOpcConfigCompleted();}" );
```

See Also

Section *Attributes* on page 26

int setServerFlags([char* | int] serverName, char* flagsString, int flags = 0)

Updates the server's *flags* attribute. This controls whether the OPC client is asking the **client** or the **server** or **both** for information. The flagsString can contain the settings for: **errorstrings** – where the client asks for error strings to use in error messages. **propertydescriptions** – where the client gets the description strings for the properties from. The default value is "errorstrings=server, propertydescriptions=server", meaning that the client asks the server for this information. When specifying **both**, the client first checks if it is has some useful information, and if it does not, then it asks the server for the information. When specifying **client**, the server will not be asked for this information at all, and the information will contain the word 'unknown' instead. This attribute is normally only set to get some useful information even when the server has a bad implementation of the `IServer->GetErrorString` function or the `IOPCItemProperties->QueryAvailableProperties` function.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flagsString – The new value of the *flags* attribute in the server.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

Example

```
int sts = setServerFlags( "MyServer",
              "errorstrings=client, propertydescriptions=both" );
```

See Also

Section *Attributes* on page 26

---

int setServerNotifyCommand([char* | int] serverName, char* notifyCommand, int flags = 0)

Updates the server's *connectNotify* attribute. The notify command is pTALK code to be executed when the connection to the OPC server is established, broken or terminated.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29.

notifyCommand – The new pTALK code for the *connectNotify* attribute.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

Example

```
int sts = setServerNotifyCommand( "MyServer",
                                  "{OpcConnectionChanged();}" );
```

See Also

Section *Attributes* on page 26

---

int setServerReconnectAfterShutdown([char* | int] serverName, int reconnect, int flags = 0)

Updates the server's *reconnectAfterShutdown* attribute. If argument *reconnect* is true (=1), the OPC plugin will try to reconnect to the OPC server after receiving a shutdown message from the server. If *reconnect* is false (=0), the OPC plugin will only try to reconnect if the connection was broken.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29.

reconnect – Boolean value, 0 = false, 1 = true. Specifies whether the plugin should try to reconnect after receiving a shutdown message from the server.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

See Also

Section *Attributes* on page 26

---

int setServerReconnectInterval([char* | int] serverName, int seconds, int flags = 0)

Updates the server's *reconnectInterval* attribute.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

seconds – Number of seconds between each time the OPC plugin should try to reconnect to the server. 0 means don't try to reconnect.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

See Also

Section *Attributes* on page 26

Group functions

The following functions are used to configure groups.

Function	Description
addGroup	Adds a group to the specified server.
getGroupDeadBand	Gets the value of the group's deadBand attribute.
getGroupName	Gets the group's name
getGroupRealUpdateRate	Gets the group's actual update rate
getGroupStatus	Gets the status of a group and its items and itemProperties.
getGroupStatusChanged	Checks if there has been any changes of the group status.
getGroupUpdateCommand	Gets the value of the group's updateCommand attribute.
getGroupUpdateRate	Gets the value of the group's updateRate attribute.
getGroupUpdateType	Gets the value of the group's updateType attribute.
getMaxGroupIndex	Gets the maximum group index for the specified server.
findGroupIx	Finds group index from group name.
removeGroup	Removes the specified group from the configuration.
setGroupDeadBand	Sets a new deadBand value for the group.
setGroupUpdateCommand	Sets the group's updateCommand attribute.
setGroupUpdateRate	Sets the requested update rate for the specified group.
setGroupUpdateType	Sets the group's updateType attribute.

F u n c t i o n s :

---

int addGroup([char* | int] serverName,
char* groupName,
int updateType=0,
int updateRate=1000,
double deadBand=0,
int flags = 0)

Adds a group to the specified server. The function *applyOPCChanges* must be called after adding a group to effectuate the changes.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – The new group's name.

updateType – Normal=0, Poll=1, Async=2, Sync=3. Ref section *Attributes* on page 26

updateRate – Update rate in milliseconds, valid only if *updateType* is normal or poll.

deadBand – DeadBand. Valid only if *updateType* is normal

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

---

double getGroupDeadBand([char* | int] serverName,
[char* | int] groupName,
int flags = 0)

Returns the value of the group's *deadBand* attribute.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the value of the group's *deadBand* attribute

See Also

Section *Attributes* on page 26.

---

char* getGroupName([char* | int] serverName,
[char* | int] groupName,
int flags = 0)

Returns the group's name.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29. Normally an index.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the group's name, null pointer on failure.

---

int getGroupRealUpdateRate([char* | int] serverName, [char* | int] groupName, int flags = 0)

Returns the group's actual update rate (accepted by the OPC server) in milliseconds. This function is only relevant for groups with *updateType* = normal or poll.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the actual update rate, in milliseconds.

See Also

Section *Attributes* on page 26

---

int getGroupStatus([char*|int] serverName, [char*|int] groupName, int* numItemsOk=0, List* errorList=0, char* itemErrorTemplate=0, char* itemPropertiesErrorTemplate=0, int flags = 0)

Returns a value different from 0 if the group is connected to the OPC server, if not connected 0 is returned. If the output parameter *numItemsOk* is set, the number of items without errors is reported, if the *itemPropertiesErrorTemplate* is specified, the number of ok item properties is also part of the *numItemsOk* value. If the *errorList* is set, and *itemErrorTemplate* and *itemPropertiesErrorTemplate* is not set, the *errorList* is filled with the integer indexes of items with error (item properties are not checked). If *itemErrorTemplate* is set, the item errors are formatted according to this template, with one string for each error item into the *errorList*. If *itemPropertiesErrorTemplate* is also set, errors in item properties are also checked, and formatted according to this template and added to the *errorList*. *itemErrorTemplate* and *itemPropertiesErrorTemplate* can contain the following replacement codes: %I = OpcItemId, %E = hexadecimal error code, %D = error description, %i = configuration index of item, %ni = configuration name of item. *itemPropertiesErrorTemplate* can in addition contain the following replacement codes: %P = OpcProp integer, %p = configuration index of item property, %np = configuration name of item property.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

numItemsOk – Output-parameter for number of OK items in the group, may include item properties.

errorList – Pointer to a List object that receives the text for the items and item properties with errors.

itemErrorTemplate – Template for item error messages.

itemPropertiesErrorTemplate – Template for item property error messages.

flags – Specifies how to interpret the serverName argument. See General parameter descriptions on page 29.

Return Value

Returns 0 if the group is not connected.

---

int getGroupStatusChanged([char*|int] serverName, [char*|int] groupName, int flags = 0)

Returns a value different from 0 if errors in the group have changed, since last time this function was called for the group specified. This function can be called before *getGroupStatus*, to check if there is any change in the status, so that the total list of item and item property errors is not fetched at each update, when there is no change.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the serverName argument. See General parameter descriptions on page 29.

Return Value

Returns 0 when there has been no change since last time getGroupStatusChanged was called. If there have been no errors for the items or item properties in this group, the first time it is called, 0 is also returned.

---

char* getGroupUpdateCommand([char* | int] serverName, [char* | int] groupName, int flags = 0)

Returns the value of the group's *updateCommand* attribute.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the value of the group's *updateCommand* attribute.

See Also

Section *Attributes* on page 26.

---

int getGroupUpdateRate([char* | int] serverName, [char* | int] groupName, int flags = 0)

Returns the value of the group's *updateRate* attribute. Note that the OPC server may choose to update the group's items at a slower rate than the *updateRate* attribute. The function getGroupRealUpdateRate gives the update rate that the OPC server is really using.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the value of the group's *updateRate* attribute.

See Also

Function *getGroupRealUpdateRate*, and Section *Attributes* on page 26.

---

int getGroupUpdateType([char* | int] serverName, [char* | int] groupName, int flags = 0)

Returns the value of the group's *updateType* attribute.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the update type: 0=normal, 1=poll, 2=async, 3=sync.

See Also

Section *Attributes* on page 26.

---

int getMaxGroupIndex([char* | int] serverName, int flags = 0)

Returns the maximum group index for the specified server. Removed groups may be included in the range 1..max.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the maximum group index, -1 on failure.

Example

```
int maxG = OPC.getMaxGroupIndex( "MyServer" );
for ( int g = 1; g <= maxG; g++ )
{
   if ( OPC.findGroupIx( "MyServer", g ) >= 1 )
   {
      char* groupName = OPC.getGroupName( "MyServer", g );
      printf( "%s", groupName );
   }
}
```

See Also

General parameter descriptions on page 29

---

int findGroupIx([char* | int] serverName, [char* | int] groupName, int flags = 0)

Returns the group index. Gives no error messages if group not found.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the group index (>= 1) if the group is found, returns <= 0 if not found.

---

int removeGroup([char* | int] serverName, [char* | int] groupName, int flags = 0)

Removes the specified group from the configuration. The group is removed from the OPC server directly after this function has been called.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

---

int setGroupDeadBand([char* | int] serverName,
[char* | int] groupName,
double deadBand,
int flags = 0)

Sets a new deadBand value for the group. The function *applyOPCChanges* must be called afterwards to inform the OPC server.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

deadBand – Deadband value, 0 means no deadband

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

See Also

Section *Attributes* on page 26

---

int setGroupUpdateCommand([char* | int]serverName,
[char* | int] groupName,
char* pTalkUpdCmd,
int flags = 0)

Sets a new value for the group's *updateCommand* attribute.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

pTalkUpdCmd – The new pTALK code for *updateCommand*.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

Example

```
int status = setGroupUpdateCommand( "MyServer", "MyGroup",
                                    "{OnOpcUpdate();}" );
```

See Also

Section *Attributes* on page 26

---

int setGroupUpdateRate([char* | int] serverName,
[char* | int] groupName,
int updateRate,
int flags = 0)

Changes the requested update rate for the specified group. The function *applyOPCChanges* must be called afterwards to inform the OPC server. The OPC server may reject the requested update rate, the actual update rate can only be obtained using the function *getGroupRealUpdateRate*.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

updateRate – Requested update rate in milliseconds

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

See Also

Section *Attributes* on page 26

---

int setGroupUpdateType([char* | int] serverName,
[char* | int] groupName,
int updateType,
int flags = 0)

Changes the group's *updateType* attribute. The function *applyOPCChanges* must be called afterwards to inform the OPC server.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

updateType – Normal=0, Poll=1, Async=2, Sync=3

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

See Also

Section *Attributes* on page 26

Item functions

The following functions are used to configure items.

Function	Description
addItem	Adds an item to the specified group.
getItemName	Gets the item's name.
getItemOpcId	Returns the OPC item id.
getItemQualityAddress	Gets the name of the ProcSee variable that receives the item's quality.
getItemTimeAddress	Gets the name of the ProcSee variable that receives the item's update time.
getItemValueAddress	Gets the name of the ProcSee variable that receives the item's value.
getItemValueTypeOverride	Gets the type override used instead of the type of the variable that receives the item's value, when requesting values from the OPC server.
getMaxItemIndex	Gets the maximum item index for the specified group.
findItemIx	Finds item index from item name.
removeItem	Removes the specified item from the configuration
renameItem	Sets a new item name.
setItemQualityAddress	Sets the name of the ProcSee variable that will receive the item's quality.
setItemTimeAddress	Sets the name of the ProcSee variable that will receive the item's update time.
setItemValueAddress	Sets the name of the ProcSee variable that will receive the item's value.
setItemValueTypeOverride	Sets the type override to use when requesting values from the OPC server (the vt attribute of the item).

F u n c t i o n s :

---

int addItem([char* | int] serverName,
[char* | int] groupName,
char* itemName,
char* itemOpcId,
char* valueAddress,
char* qualityAddress=0,
char* timeAddress=0,
int flags = 0)

Adds an item to the specified group. The function *applyOPCChanges* must be called after adding an item to effectuate the changes.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – The new item's name. A null pointer creates an anonymous item

itemOpcId – The name of the item in the OPC server (UTF8 encoded).

valueAddress – The name of a ProcSee variable or attribute which will receive the value of the OPC item when updated. The item value will be converted to the ProcSee data type.

qualityAddress – The name of a ProcSee integer variable or attribute which will receive the quality of the OPC item.

timeAddress – The name of a ProcSee integer variable or attribute which will receive the update time of the OPC item. The time is converted to seconds since 1970 just like any other time used in ProcSee.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

Example

```
int status = addItem("MyServer",
                     "MyGroup",
                     "MyItemName3",
                     "itemOpcId1",
                     "MyProcess.Var1.v",
                     "MyProcess.Var1.q",
                     "MyProcess.Var1.t");
```

---

char* getItemName([char* | int] serverName, [char* | int] groupName, [char* | int] itemName, int flags = 0)

Returns the item's name.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29. Normally an index or an OPC item id.

flags – Specifies how to interpret the *serverName* and *itemName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the item's name, null pointer on failure.

---

char* getItemOpcId([char* | int] serverName, [char* | int] groupName, [char* | int] itemName, int flags = 0)

Returns the OPC item id.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *itemName* arguments. See General parameter descriptions on page 29.

Return Value

Returns the item's OPC id (UTF8 encoded), null pointer on failure

---

char* getItemQualityAddress([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
int flags = 0)

Returns the name of the ProcSee variable or attribute which will receive the quality of the updated item.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *itemName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the name of the ProcSee variable or attribute, null pointer on failure.

---

char* getItemTimeAddress([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
int flags = 0)

Returns the name of the ProcSee variable or attribute which will receive the update time of the updated item.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns the name of the ProcSee variable or attribute, null pointer on failure.

---

char* getItemValueAddress([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
int flags = 0)

Returns the name of the ProcSee variable or attribute which will receive the item value from the OPC server.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns the name of the ProcSee variable or attribute, null pointer on failure.

---

char* getItemValueTypeOverride([char* | int] serverName, [char* | int] groupName, [char* | int] itemName, int flags = 0)

Returns the override type, used when requesting the value from the OPC server, instead of the type of the ProcSee variable. If the override is not set, it returns an empty string.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29.

groupName – Identifies the group. See *General parameter descriptions* on page 29.

itemName – Identifies the item. See *General parameter descriptions* on page 29.

flags – Specifies how to interpret the *serverName* and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns an empty string (“”) if no override, or a string like “R8” or “ARRAY|R4”. Returns a null pointer on failure.

---

int getMaxItemIndex([char* | int] serverName, [char* | int] groupName, int flags = 0)

Returns the maximum item index for the specified group. Removed items may be included in the result.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the max item index, -1 on failure.

Example

```
int n = OPC.getMaxItemIndex( “MyServer”, “GroupName” );
```

```
for( int i=1 ; i<=n; i++ )
{
      if ( OPC.findItemIx( "MyServer", "GroupName", i ) >= 1 )
      {
           char* itemName = OPC.getItemName( "MyServer",
                                             "GroupName", i );
           printf( "%s", itemName );
      }
}
```

---

int findItemIx([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
int flags = 0)

Returns the item index. Gives no error messages if item not found.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns the item index (>= 1) if the item is found, returns <= 0 if not found.

---

int removeItem([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
int flags = 0)

Removes the specified item from the configuration.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns 0 on failure, and non-zero on success.

---

int renameItem([char*|int] serverName,
[char*|int] groupName,
[char*|int] itemName,
char* newName,
int flags = 0)

Sets a new item name.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

newName – the new item name

flags – Specifies how to interpret the *serverName* and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns 0 on failure, and non-zero on success.

---

int setItemQualityAddress([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
char* qualityAddress,
int flags = 0)

Sets the name of the ProcSee variable or attribute which will receive the item's quality field from the OPC server.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

qualityAddress – The name of the ProcSee integer variable or attribute.

flags – Specifies how to interpret the *serverName* and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns 0 on failure, and non-zero on success.

---

int setItemTimeAddress([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
char* timeAddress,
int flags = 0)

Sets the address of a ProcSee variable which will be updated with the time field when normal or poll updates are received from the OPC server.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

timeAddress – The name of the ProcSee integer variable or attribute. The time is converted to seconds since 1970 just like any other time used in ProcSee.

flags – Specifies how to interpret the *serverName* and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns 0 on failure, and non-zero on success.

---

int setItemValueAddress([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
char* valueAddress,
int flags = 0)

Sets the name of the ProcSee variable or attribute which will be updated when normal or poll updates are received from the OPC server. The type of this ProcSee variable or attribute should be double, float, short int, int, or char*, or an array of or pointer to one of these types.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

valueAddress – The name of the ProcSee variable or attribute. The received value will be converted to the type of this ProcSee variable or attribute.

flags – Specifies how to interpret the *serverName* and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns 0 on failure, and non-zero on success.

---

int setItemValueTypeOverride([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
char* vt,
int flags = 0)

Sets the type override to use instead of the type of the ProcSee variable specified with the setItemValueAddress function, when requesting values from the OPC server (the vt attribute of the item). This function should normally never be used, since the OPC server normally does a good job of converting the value to the needed type in the cases where any conversion is needed. If this type override is set, the OPC server will try to send a value with the type specified in this override, and then the ProcSee OPC plugin will convert the received value to the type of the ProcSee variable.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29.

groupName – Identifies the group. See *General parameter descriptions* on page 29.

itemName – Identifies the item. See *General parameter descriptions* on page 29.

vt – A string specifying the override type. If the string is empty, no override is set. Examples of other values are “I4”, “R4”, “R8”, “ARRAY|R4”.

flags – Specifies how to interpret the *serverName* and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns 0 on failure, and non-zero on success.

Item Property functions

The following functions are used to configure item properties

Function	Description
addItemProperty	Adds an item property to the specified item.
getItemPropertyAddress	Gets the name of the ProcSee variable that receives the property value.
getItemPropertyName	Gets the item property’s name.
getItemPropertyOpcProp	Gets the item property’s OPC property number.
getMaxItemPropertyIndex	Gets the item’s maximum item property index.
findItemPropertyIx	Finds item property index from item property name.
removeItemProperty	Removes the item property.
renameItemProperty	Sets a new name for the item property.
setItemPropertyAddress	Sets the name of the ProcSee variable that will receive the property value.

F u n c t i o n s :

---

int addItemProperty([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
char* propertyName,
int propertyOpcId,
char* valueAddress,
int flags = 0)

Adds an item property to the specified item. The function *applyOPCChanges* must be called after adding an item property to effectuate the changes.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

propertyName – The new item property's name. A null pointer creates an anonymous item property.

propertyOpcId – The propertyOpcId is the number used to access the property on the OPC server.

valueAddress – The name of the ProcSee variable or attribute which receives the value of this property.

flags – Specifies how to interpret the *serverName*, and *itemName* arguments. *See General parameter descriptions on page 29*.

Return Value

Returns 0 on failure, and non-zero on success.

---

char* getItemPropertyAddress([char* | int] serverName, [char* | int] groupName, [char* | int] itemName, [char* | int] propertyName, int flags = 0)

Returns the name of the ProcSee variable or attribute which will receive the property value.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

propertyName – Identifies the item property. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName*, *itemName*, and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the name of the ProcSee variable or attribute, null pointer on failure.

---

char* getItemPropertyName([char* | int] serverName, [char* | int] groupName, [char* | int] itemName, [char* | int] propertyName, int flags = 0)

Returns the item property's name.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

propertyName – Identifies the item property. See *General parameter descriptions* on page 29. Normally an index or property number.

flags – Specifies how to interpret the *serverName*, *itemName*, and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the name of the item property, null pointer on failure.

---

int getItemPropertyOpcProp([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
[char* | int] propertyName,
int flags = 0)

Returns the OPC property number for the specified item property.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

propertyName – Identifies the item property. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName*, *itemName*, and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the OPC property number for the specified item property, 0 on failure.

---

int getMaxItemPropertyIndex([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
int flags = 0)

Returns the maximum item property index for the specified item. Removed item properties may be included in the result.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *itemName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the maximum item property index, -1 on failure.

Example

```
int n = OPC.getMaxItemPropertyIndex(“MyServer”,
                                    “MyGroup”,
                                    “MyItem”);
for (int i=1; i<=n; i++)
{
  if(OPC.findItemPropertyIx(“MyServer”,”MyGroup”,”MyItem”,i) >= 1)
  {
```

```
    char* propName = OPC.getItemPropertyName( "MyServer",
                                              "MyGroup",
                                              "MyItem",
                                              i );
    printf( "%s", propName );
  }
}
```

---

int findItemPropertyIx([char* | int] serverName, [char* | int] groupName, [char* | int] itemName, [char* | int] propertyName, int flags = 0)

Returns the item property index. Gives no error messages if item property not found.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

propertyName – Identifies the item property. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName*, *itemName*, and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the item property index (>= 1) if the item property is found, returns <= 0 if not found.

---

int removeItemProperty([char* | int] serverName, [char* | int] groupName, [char* | int] itemName, [char* | int] propertyName, int flags = 0)

Removes the specified item property from the configuration.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

propertyName – Identifies the item property. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName*, *itemName*, and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, and non-zero on success.

---

int renameItemProperty([char*|int] serverName,
[char*|int] groupName,
[char*|int] itemName,
[char*|int] propertyName,
char* newName,
int flags = 0)

Sets a new item property name.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

propertyName – Identifies the item property. See *General parameter descriptions* on page 29

newName – The new item property name

flags – Specifies how to interpret the *serverName*, *itemName*, and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, and non-zero on success.

---

int setItemPropertyAddress([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
[char* | int] propertyName,
char* valueAddress,
int flags = 0)

Sets the name of the ProcSee variable or attribute that will receive the item property value.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

propertyName – Identifies the item. See *General parameter descriptions* on page 29

valueAddress – The name of the ProcSee variable or attribute.

flags – Specifies how to interpret the *serverName*, *itemName* and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, and non-zero on success.

Property functions

The following functions are used to configure properties.

Function	Description
addProperty	Adds a property to the specified server.
getPropertyAddress	Gets the name of the ProcSee variable which receives the property's value.
getPropertyName	Gets the property's name.
getPropertyOpcItemId	Gets the property's OPC item id
getPropertyOpcProp	Gets the property's OPC property number.
getMaxPropertyIndex	Gets the maximum property index for the specified server.
findPropertyIx	Finds property index from property name.
removeProperty	Removes the specified property from the configuration.
renameProperty	Set a new property name.
setPropertyAddress	Sets the name of the ProcSee variable which will receive the property's value.

F u n c t i o n s :

---

int addProperty([char* | int] serverName,
char* propertyName,
char* opcItemId,
int propertyOpcId,
char* valueAddress,
int flags = 0)

Adds a property to the specified server. The function *applyOPCChanges* must be called afterwards to effectuate the changes.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

propertyName – The new property's name. A null pointer creates an anonymous property.

opcItemId – The OPC item id (UTF8 encoded) that has this OPC property.

propertyOpcId – The propertyOpcId is the number used to access the property on the OPC server.

valueAddress – The name of the ProcSee variable or attribute which will receive the property's value.

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, and non-zero on success.

---

char* getPropertyAddress([char* | int] serverName,
[char* | int] propertyName,
int flags = 0)

Gets the name of the ProcSee variable or attribute which receives the property's value.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

propertyName – Identifies the property. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the name of the ProcSee variable or attribute, null pointer on failure.

Example

```
char* address = getPropertyAddress("MyServer", "MyProperty");
```

---

char* getPropertyName([char* | int] serverName, [char* | int] propertyName, int flags = 0)

Returns the property's name.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

propertyName – Identifies the property. See *General parameter descriptions* on page 29. Normally an index or property number.

flags – Specifies how to interpret the *serverName* and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the name of the property, null pointer on failure.

See Also

For more information see, getMaxPropertyIndex(…)

---

char* getPropertyOpcItemId([char* | int] serverName, [char* | int] propertyName, int flags = 0)

Returns the OPC item id for the specified property.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

propertyName – Identifies the property. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the OPC item id, UTF8 encoded. Returns null pointer on failure.

---

int getPropertyOpcProp([char* | int] serverName, [char* | int] propertyName, int flags = 0)

Returns the OPC property number for the specified property.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

propertyName – Identifies the property. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the OPC property number, 0 on failure.

---

int getMaxPropertyIndex([char* | int] serverName, int flags = 0)

Returns the maximum property index for the specified server. Removed properties may be included in the result.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns the number of properties configured in the server, -1 on failure.

Example

```
int n = OPC.getMaxPropertyIndex( "MyServer" );
for ( int i=1; i<=n; i++ )
{
  if ( OPC.findPropertyIx( "MyServer", i ) >= 1 )
  {
       char* propertyName = OPC.getPropertyName( "MyServer", i );
       printf( "%s", propertyName );
  }
}
```

---

int findPropertyIx([char* | int] serverName,
[char* | int] propertyName,
int flags = 0)

Returns the property index. Gives no error messages if property not found.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

propertyName – Identifies the property. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns the property index (>= 1) if the property is found, returns <= 0 if not found.

---

int removeProperty([char* | int]serverName,
[char* | int] propertyName,
int flags = 0)

Removes the specified property from the configuration.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

propertyName – Identifies the property. See *General parameter descriptions* on page 29

flags – Specifies how to interpret the *serverName* and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, and non-zero on success.

---

int renameProperty([char*|int] serverName,
[char*|int] propertyName,
char* newName,
int flags = 0)

Set a new property name.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

propertyName – Identifies the property. See *General parameter descriptions* on page 29

newName – The new property name

flags – Specifies how to interpret the *serverName* and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, and non-zero on success.

---

int setPropertyAddress([char* | int] serverName,
[char* | int] propertyName,
char* valueAddress,
int flags = 0)

Sets the name of the ProcSee variable or attribute which will receive the property's value.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29

propertyName – Identifies the property. See *General parameter descriptions* on page 29

valueAddress – The name of the ProcSee variable or attribute.

flags – Specifies how to interpret the *serverName* and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, and non-zero on success.

Runtime functions

The following functions do not change the configuration. This category covers functions for enabling or disabling OPC group subscription, write functions and property functions.

Function	Description
enableGroup	Enables or disables the specified group
writeItemValue	Writes a value to the specified OPC item.
updateItemPropertyValue	Updates an item property value
updatePropertyValue	Updates a property value

F u n c t i o n s :

---

int enableGroup([char* | int] serverName,
[char* | int] groupName,
int enable,
int flags = 0)

Enables or disables the specified group(s). Functionality for enabling and disabling groups can be relevant in situations where there is a one-to-one relationship between a group and a picture. When a picture is displayed or undisplayed, the corresponding group can be enabled/disabled.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29. If *serverName* is an empty string, *groupName* is searched for in all configured servers.

groupName – Identifies the group. See *General parameter descriptions* on page 29. If *groupName* is an empty string, all groups in the specified server(s) are enabled / disabled.

enable – Enable=1, disable=0;

flags – Specifies how to interpret the *serverName* argument. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

---

int writeItemValue([char* | int] serverName,
[char* | int] groupName,
[char* | int] itemName,
any newValue,
int flags = 0)

Writes *a value* to the specified OPC item.

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29.

groupName – Identifies the group. See *General parameter descriptions* on page 29

itemName – Identifies the item. See *General parameter descriptions* on page 29

newValue – The value to write to the OPC item. If the OPC item is a one dimensional array, the value should be an array or a pointer to array-elements, that can be created with for instance the pTalk function newArray.

flags – Specifies how to interpret the *serverName* and *itemName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, non-zero on success.

---

int updateItemPropertyValue([char* | int] serverName, [char* | int] groupName, [char* | int] itemName, [char* | int] propertyName, int flags = 0)

Updates item property value(s).

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29. If *serverName* is an empty string, all servers are updated.

groupName – Identifies the group. See *General parameter descriptions* on page 29. If *groupName* is an empty string, all groups are updated.

itemName – Identifies the item. See *General parameter descriptions* on page 29. If *itemName* is an empty string, all items are updated.

propertyName – Identifies the item property. See *General parameter descriptions* on page 29. If *propertyName* is an empty string, all item properties are updated.

flags – Specifies how to interpret the *serverName*, *itemName*, and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, and non-zero on success.

---

int updatePropertyValue([char* | int] serverName, [char* | int] propertyName, int flags = 0)

Updates property value(s).

Parameters

serverName – Identifies the server. See *General parameter descriptions* on page 29. If *serverName* is an empty string, all servers are updated.

propertyName – Identifies the property. See *General parameter descriptions* on page 29. If *propertyName* is an empty string, all properties are updated.

flags – Specifies how to interpret the *serverName* and *propertyName* arguments. See *General parameter descriptions* on page 29.

Return Value

Returns 0 on failure, and non-zero on success.

Index

U

W