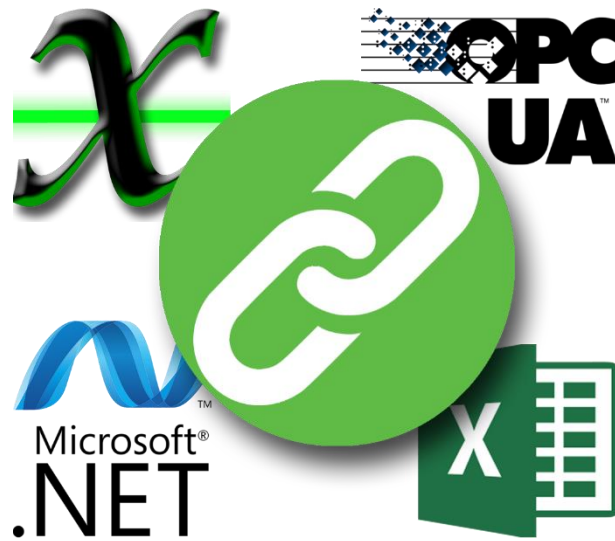


PEAXACT Application Server User Manual

Version 4.7
2019-03-22



S•PACT GmbH
Burtscheider Str. 1
52064 Aachen
Germany

phone: +49 241 9569 9812
fax: +49 241 4354 4308
e-mail: support@s-pact.de

© COPYRIGHT 2019 by S-PACT GmbH

The software described in this document is furnished under a license agreement. The software may be used only under the terms of the license agreement.

Software is based on MATLAB®. © 1984-2019 The MathWorks, Inc.

CONTENTS

Contents	3
1 Quick Start	4
1.1 What is PEAXACT Application Server?	4
1.2 Getting Help	4
1.3 Installation & License Activation	5
1.4 Before You Start	8
2 Application Programming Interface (API)	9
2.1 .NET API	9
2.2 COM API	20
2.3 Programming Examples	25
3 Custom Interfaces	29
3.1 OPUS Process	29
3.2 HoloPro	32
4 PEAXACT ProcessLink	36
4.1 Setup	36
4.2 Real-time Table	38
4.3 Real-time Chart	38
4.4 Result File	39
4.5 OPC UA Server	40
4.6 OPC Security Setup	41
4.7 Command Line Usage	41
4.8 Additional Notes	42
5 Trouble Shooting	43

1 QUICK START

1.1 What is PEAXACT Application Server?

The PEAXACT Application Server gives third-party applications access to PEAXACT analysis methods by means of an [application programming interface](#) (API). The API is available as

- .NET assembly
- COM component (Component Object Model)

Any application supporting one of these standards will be able to programmatically integrate PEAXACT as a back-end analyzer.

In addition, the Application Server provides ready-to-use [custom interfaces](#) for

- OPUS Process from Bruker
- HoloPro from Kaiser Optical Systems

as well an interactive Windows App – the [PEAXACT ProcessLink](#) – to link PEAXACT data analysis to any third-party application without the need to write custom software at all.

1.2 Getting Help

User Manual

This user manual documents a certain version of the PEAXACT Application Server. You can find the version number and release date on the title page.

We are continuously working on improving the manual. The latest document version is distributed as PDF file with each PEAXACT software update. The file is located in subdirectory `help` of the PEAXACT installation directory.

Technical Support

The Technical Support can be contacted in different ways:

- E-mail to support@s-pact.de
- Web form at <http://www.s-pact.com/support>

Note: A subscription of S•PACT Software Maintenance Service (SMS) is required to be eligible for technical support. The first year of SMS is included with new PEAXACT product licenses.

Blog

The PEAXACT Blog was launched as a free source of information complementary to the user manual. It contains tutorials, how-tos, and tips & tricks.

See: <http://www.s-pact.com/blog>

1.3 Installation & License Activation

1.3.1 System Requirements

- Microsoft Windows 7 SP1 or later (32-bit or 64-bit)
- Any Intel or AMD x86 processor with SSE2 support (2 GHz recommended)
- 2 GB RAM (4 GB recommended)
- Microsoft .NET Framework 4.5 or later

1.3.2 Licensing

PEAXACT software is furnished under a license agreement. The software may be used only under the terms of the license agreement.

The PEAXACT Application Server can be installed and operated on a given number of designated computers, provided it is only operated locally (i.e. not remotely). The number of simultaneous users is not limited. For the full and legally valid conditions please refer to the license agreement document.

1.3.3 Installation

Step 1: Before You Install

- Make sure your computer fulfills the system requirements.
- When upgrading an existing installation, visit <http://www.s-pact.com/peaxact/whatsnew> and read the upgrade notes.
- Make sure you have administrator privileges to perform the installation.
- Make sure your license is valid for the major version number. If you do not have a license yet you can get a free trial license or purchase a license after installation.

Note: The PEAXACT version consists of two numbers

<major version>.<minor version >, e.g. 4.5

Step 2: Install PEAXACT

- Download the PEAXACT Installer from <http://www.s-pact.com/peaxact/download>

Note: The installer's filename is `peaxactInstaller_<version>_<platform>.exe`
<version> is the version number; <platform> is either `win32` or `win64`. The 32-bit version also runs on 64-bit platforms, but not vice versa.

Note: Different major versions can be installed side-by-side, e.g. versions 3 and 4.

Note: The installer upgrades earlier installations of the same major version.

Online Installation (Web Installation)

- If you are going to install PEAXACT on a computer which is connected to the internet you do not need to download any additional files.
- Run the PEAXACT Installer and follow the setup instructions. Additional runtime packages are downloaded and installed automatically if detected missing.

Offline Installation

- If you are planning to install PEAXACT on a computer without internet access you have to download additional runtime packages in advance from <http://www.s-pact.com/peaxact/runtime>
- Make sure to download runtime packages for the same platform as the PEAXACT installer (32-bit or 64-bit)
- Save all installer files to a folder on your hard drive / flash drive. Do not rename files.
- Run the PEAXACT Installer file from this folder and follow the setup instructions. Runtime packages are installed automatically if detected missing.

Step 3: After Installation

- After a new product installation continue with [License Activation](#).
- After upgrading an existing installation check the upgrade notes at <http://www.s-pact.com/peaxact/whatsnew> for further upgrade steps.

1.3.4 License Activation

License activation involves loading a valid license file.

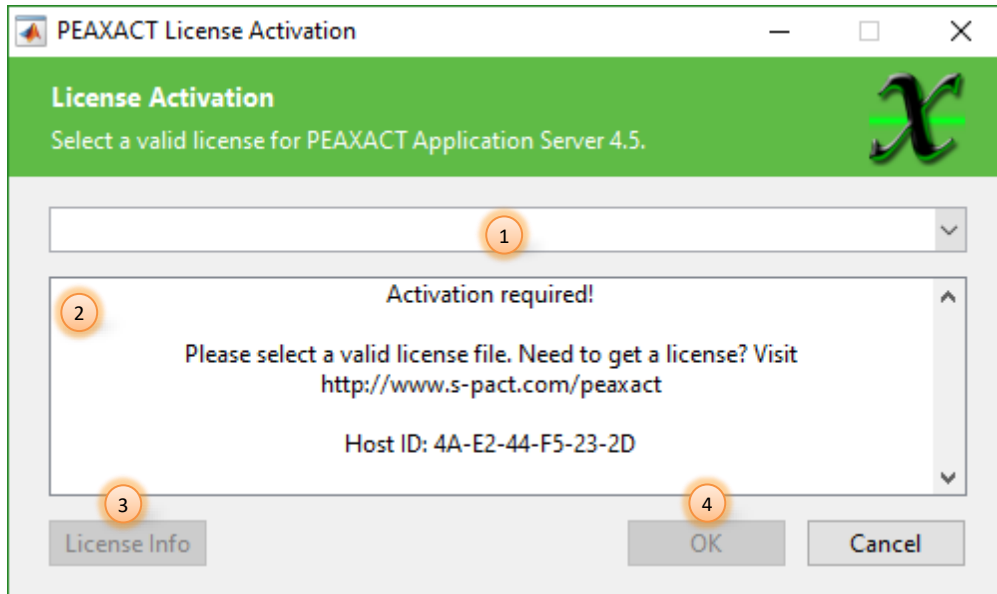
Step 1: Get your license file

- If you already have a license file continue with [step 2](#).
- To get a **trial license**, visit <http://www.s-pact.com/peaxact>. Once you have received your license file continue with [step 2](#).
- For **purchased licenses**, licenses activation associates the use of PEAXACT with designated computers by means of a Host ID. The Host ID is a MAC address or the serial number of volume c of the computer PEAXACT is installed on.
 - From the Windows start menu select **Programs > PEAXACT 4 > Activate PEAXACT Application Server**
 - Wait until the License Activation Dialog is displayed
 - Copy the Host ID from the dialog window and send it to support@s-pact.de
 - Click Cancel for now. Once you've received your license file continue with [step 2](#).

Note: You can also type `getmac` at the command prompt and use the first MAC address as Host ID.

Step 2: Activate license

- Click the Windows start menu and select **Programs > PEAXACT 4 > Activate PEAXACT Application Server**
- Wait until the License Activation Dialog is displayed



License Activation Dialog

- | | |
|--------------------------|------------------------------------|
| (1) License selection | (3) Additional license information |
| (2) Status of activation | (4) Apply and close |

- Choose **Import License...** from the list (1) to browse for a valid license file. If the license is valid the license file is copied to the license directory.
- Once a valid license is selected, you can click on the **License Info** button (3) to learn more about the license or on the **OK** button (4) to accept the selection.

Per-machine license vs. per-user license

If you perform the activation with administrator privileges, licenses will be activated per-machine, i.e. for all Windows users. Otherwise, licenses will be activated per-user, i.e. for the logged-on user. Per-machine licenses take precedence over per-user licenses. Once a per-machine license is activated the License Activation Dialog gets locked for regular users.

1.4 Before You Start

Before you access the Application Server for the first time you should test whether everything is installed correctly by running a *diagnosis program*. Click the Windows start menu and select **Programs > PEAXACT 4 > Diagnosis Tool for PEAXACT Application Server**.

The diagnosis program performs some tests and suggests possible solutions in case of problems. You should fix all problems before you proceed. Typical problems include:

- MATLAB Compiler Runtime (MCR) is not installed correctly
- Required DLL files are not registered correctly
- Platform-dependent problems (e.g. running 32-bit software instead of 64-bit)

You could run the diagnosis program at any time to check whether the interface still works correctly and to reveal possible errors.

Note: During the diagnosis, you may be prompted to [activate a license](#).

2 APPLICATION PROGRAMMING INTERFACE (API)

2.1 .NET API

The **.NET API** is a set of classes contained in a design-time assembly you would compile and link against when building your own managed assemblies. Before you can use it, you need to reference the assembly in your Visual Studio project. In Visual Studio, right-click on a project, for example, and click "Add References...". The assembly file is located at

```
<INSTALLPATH>\DLL\PeaxactApplicationServer.dll
```

Note: The PEAXACT Application Server requires Microsoft .NET Framework 4.5 or later to be installed on your computer.

Backward Compatibility

New versions of the .NET API will be backward compatible. Therefore, it is highly recommended that you do not bind your application to a specific version of the Application Server assembly but instead bind dynamically to the newest version installed on the target computer. See Section 2.3.3 for a programming example.

Support for Asynchronous Analyses

The .NET API provides methods to run analyses asynchronously in a background thread. However, note that analyses are executed internally by the MATLAB Runtime which is single-threaded, i.e. if you run multiple asynchronous operations in parallel they will still be executed one after another.

Exception Handling

The .NET API throws exceptions of predefined .NET Framework exception types with specific error messages. Use exception handling code (try/catch blocks) appropriately for all method calls.

2.1.1 Namespaces

The .NET API contains two namespaces, both providing types with identical names and similar functionality.

Types of the **S_PACT.PEAXACT**-namespace are type-safe. Use these types in your Visual Studio projects.

Types of the **S_PACT.PEAXACT.Legacy**-namespace are COM-visible. These types are not type-safe for higher compatibility with older COM clients such as VBScript. Never use these legacy-types in your Visual Studio projects; they are meant to be used by COM clients only.

2.1.2 ApplicationServer Class

The `ApplicationServer` class is the root class of the API. It provides methods to `Start()` and `Stop()` the server. Once started you can create `Session` objects with the `NewSession()` or `LoadSession()` methods. A `Session` is a container for models and data sets, representing an isolated environment where analyses execute.

Constructor

The `ApplicationServer` class has no public constructor. The class implements a singleton pattern with a static property `Instance`.

Properties

Instance : `ApplicationServer`

A static property holding a singleton instance of the `ApplicationServer` class.

IsStarted : `bool`

A flag indicating whether the Application Server has been started.

Methods

`Start()`

`Start(StartupOptions options) : void`

Starts the Application Server. The method does nothing if the server has been started already.

options: Startup options. (null = defaults)

`Stop() : void`

Stops the Application Server. The method waits until all sessions are idle before it returns. You should always call this method explicitly before exiting your application to properly clean up any unmanaged resources.

`NewSession() : Session`

Creates a new `Session` instance.

`LoadSession(string filename) : Session`

Creates a new `Session` instance by loading a PEAXACT session file.

filename: Filename of a PEAXACT session (extension PXS).

2.1.3 StartOptions Class

The `StartOptions` class provides optional settings to control the startup behavior of the Application Server.

Constructor

`StartOptions()`

Initializes a new instance of the `StartOptions` class with default properties.

Properties

LicenseFilename : string

Filename of a PEAXACT license file (extension LIC). (null or empty = detect automatically; "interactive" = show file dialog if required)

In case of auto-detection, a license file is searched for in the Windows Registry at: HKEY_LOCAL_MACHINE\S-PACT\PEAXACT Application Server 4\LicenseSource and:

HKEY_CURRENT_USER\S-PACT\PEAXACT Application Server 4\LicenseSource

LicensePassword : string

Password to unlock a protected license file. The filename of a protected license must be specified explicitly by `LicenseFilename`. Protected license files are only available upon special request.

LogFilename : string

Filename of a log file. (null or empty = no logging; "default" = default filename)

2.1.4 Session Class

A `Session` is a container for models and data sets, representing an isolated environment where analyses execute.

Constructor

The `Session` class has no public constructor. Instances of this class can only be created using the `NewSession()` or `LoadSession()` methods of the `ApplicationServer` class.

Properties

DataSetUris : string[]

Gets URIs of added data sets.

ModelFileNames : string[]

Gets filenames of added models.

IntegrationRangeNames : string[]

Gets names of integration ranges from all added models in the order models were added to the session.

ComponentNames : string[]

Gets names of non-empty hard model components from all added models in the order models were added to the session.

CalibratedFeatureNames : string[]

Gets names of calibrated features from all added models in the order models were added to the session.

Methods**Dispose() : void**

Frees the native resources associated with this object.

AddDataSet(string uri) : void**AddDataSet(string uri, bool removeExisting) : void****AddDataSet(string uri, bool removeExisting, double[] x, double[] y, params object[] z)**

Add a data set to the session.

uri: URI of a data set to be added to the session.

removeExisting: Flag indicating whether to remove all other data sets from the session before adding the new one. This is useful for single-shot analyses.

x: Array of x-values to be used instead of values read from the data file.

y: Array of y-values to be used instead of values read from the data file.

z: Optional parameter/value pairs representing additional data set features. The parameter must be a string; the value must be of type double.

AddModel(string filename) : void

Add a model to the session.

filename: Filename of a PEAXACT model (extension PXM) to be added to the session. It can also be the name of a PEAXACT session file (extension PXS) containing any number of models to be added.

RemoveDataSet(int dataSetIndex) : void**RemoveDataSet(string dataSetUri) : void**

Removes a data set from the session.

dataSetIndex: The one-based index of the data set to be removed from the session.

dataSetUri: The URI of the data set to be removed from the session.

RemoveModel(int modelIndex) : void**RemoveModel(string modelFilename) : void**

Removes a model from the session.

modelIndex: The one-based index of the model to be removed from the session.

modelFilename: The filename of the model to be removed from the session.

GetModelInfo(int modelIndex) : [ModelInfo](#)

GetModelInfo(string modelFilename) : **ModelInfo**

Gets information about a model.

modelIndex: The one-based index of the model in the session.

modelFilename: The filename of a model in the session.

AnalysisPeakPicking(int dataSetIndex) : [PeakPickingResults](#)

AnalysisPeakPicking(string dataSetUri) : **PeakPickingResults**

AnalysisPeakPicking(int dataSetIndex, double minPeakHeight) : **PeakPickingResults**

AnalysisPeakPicking(string dataSetUri, double minPeakHeight) : **PeakPickingResults**

Performs peak picking for a single data set using pretreatments of the first added model (if any).

dataSetIndex: The one-based index of a data set in the session.

dataSetUri: The URI of a data set in the session.

minPeakHeight: Minimum height for peak detection.

AnalysisIntegration() : [IntegrationResults](#)

AnalysisIntegration(int[] integrationRangeIndices) : **IntegrationResults**

AnalysisIntegration(string[] integrationRangeNames) : **IntegrationResults**

Performs peak integration for all added data sets using all added integration models.

integrationRangeIndices: One-based indices of integration ranges (as in *IntegrationRangeNames*) to be considered for the analysis.

integrationRangeNames: Names of integration ranges (as in *IntegrationRangeNames*) to be considered for the analysis.

AnalysisIntegrationAsync(int[] integrationRangeIndices, CancellationToken cancellationToken, IProgress<int> progress) : **Task<IntegrationResults>**

AnalysisIntegrationAsync(string[] integrationRangeNames, CancellationToken cancellationToken, IProgress<int> progress) : **Task<IntegrationResults>**

Asynchronously performs peak integration for all added data sets using all added integration models.

integrationRangeIndices: One-based indices of integration ranges (as in *IntegrationRangeNames*) to be considered for the analysis.

integrationRangeNames: Names of integration ranges (as in *IntegrationRangeNames*) to be considered for the analysis.

cancellationToken: The token to monitor for cancellation requests.

progress: The provider for progress updates.

AnalysisComponentFitting() : [ComponentFittingResults](#)

AnalysisComponentFitting(int[] componentIndices) : **ComponentFittingResults**

AnalysisComponentFitting(string[] componentNames) : **ComponentFittingResults**

Performs component fitting for all added data sets using all added hard models.

componentIndices: One-based indices of hard model components (as in *ComponentNames*) to be considered for the analysis.

componentNames: Names of hard model components (as in *ComponentNames*) to be considered for the analysis.

AnalysisComponentFittingAsync(int[] componentIndices, CancellationToken cancellationToken, IProgress<int> progress) : Task<[ComponentFittingResults](#)>

AnalysisComponentFittingAsync(string[] componentNames, CancellationToken cancellationToken, IProgress<int> progress) : Task<ComponentFittingResults>

Asynchronously performs component fitting for all added data sets using all added hard models.

componentIndices: One-based indices of hard model components (as in `ComponentNames`) to be considered for the analysis.

componentNames: Names of hard model components (as in `ComponentNames`) to be considered for the analysis.

cancellationToken: The token to monitor for cancellation requests.

progress: The provider for progress updates.

AnalysisPrediction() : [PredictionResults](#)

AnalysisPrediction(int[] featureIndices) : PredictionResults

AnalysisPrediction(string[] featureNames) : PredictionResults

Performs feature prediction for all added data sets using all added calibration models.

featureIndices: One-based indices of calibrated features (as in `CalibratedFeatureNames`) to be considered for the analysis.

featureNames: Names of calibrated features (as in `CalibratedFeatureNames`) to be considered for the analysis.

AnalysisPredictionAsync(int[] featureIndices, CancellationToken cancellationToken, IProgress<int> progress) : Task<[PredictionResults](#)>

AnalysisComponentFittingAsync(string[] featureNames, CancellationToken cancellationToken, IProgress<int> progress) : Task<PredictionResults>

Asynchronously performs feature prediction for all added data sets using all added calibration models.

featureIndices: One-based indices of calibrated features (as in `CalibratedFeatureNames`) to be considered for the analysis.

featureNames: Names of calibrated features (as in `CalibratedFeatureNames`) to be considered for the analysis.

cancellationToken: The token to monitor for cancellation requests.

progress: The provider for progress updates.

AnalysisMcr(int numComponents) : [McrResults](#)

AnalysisMcr(int numComponents, [McrOptions](#) options) : McrResults

Performs MCR-ALS for all added data sets using pretreatments of the first added model (if any).

numComponents: Number of unknown components to be identified from the data.

options: Additional options for the algorithm.

AnalysisMcrAsync(int numComponents, [McrOptions](#) options, CancellationToken cancellationToken, IProgress<int> progress) : Task<[McrResults](#)>

Asynchronously performs MCR-ALS for all added data sets using pretreatments of the first added model (if any).

numComponents: Number of unknown components to be identified from the data.

options: Additional options for the algorithm.

cancellationToken: The token to monitor for cancellation requests.

progress: The provider for progress updates.

AnalysisHmfa(int numComponents) : [HmfaResults](#)

AnalysisHmfa(int numComponents, [HmfaOptions](#) options) : **HmfaResults**

Performs HMFA for all added data sets using the first added model.

numComponents: Number of unknown components to be identified from the data.

options: Additional options for the algorithm.

AnalysisHmfaAsync(int numComponents, [HmfaOptions](#) options, CancellationToken cancellationToken, IProgress<int> progress) : **Task<[HmfaResults](#)>**

Asynchronously performs HMFA for all added data sets using the first added model.

numComponents: Number of unknown components to be identified from the data.

options: Additional options for the algorithm.

cancellationToken: The token to monitor for cancellation requests.

progress: The provider for progress updates.

2.1.5 McrOptions Class

The `McrOptions` class provides optional settings for the MCR-ALS analysis.

Constructor

McrOptions()

Initializes a new instance of the `McrOptions` class with default properties.

Properties

Co : double[,]

Optional 2D array of initial concentrations. Rows correspond to data sets, columns correspond to components.

If `co` is null, concentrations are initialized with previous results (if available).

If `co` is an empty Array, concentrations are initialized implicitly (reset).

ToleranceRmse : double

Criterion for stopping the MCR-ALS algorithm when progress between iterations drops below the tolerance (default = $1e-5$).

MaxIterations : int

Criterion for stopping the MCR-ALS algorithm after a maximum number of iterations (default = 100).

MaxUnsuccessfulAttempts : int

Criterion for stopping the MCR-ALS algorithm after a maximum number of unsuccessful iterations (default = 20).

IsNonnegativeC : double

Enable or disable the non-negativity constraint for component concentrations (default = false).

IsNonnegativeS : double

Enable or disable the non-negativity constraint for component spectra (default = false).

IsUnimodalC : double

Enable or disable the unimodality constraint for component concentrations (default = false).

IsClosureC : bool

Enable or disable the closure constraint for component concentrations (default = false).

2.1.6 HmfaOptions Class

The `HmfaOptions` class provides optional settings for the HMFA analysis.

Constructor

HmfaOptions()

Initializes a new instance of the `HmfaOptions` class with default properties.

Properties

IsClosureC : bool

Enable or disable the closure constraint for component concentrations (default = false).

2.1.7 ModelInfo Struct

The `ModelInfo` is a read-only structure returned by `Session.GetModelInfo()` providing information about a model. In particular, it is useful to determine the kinds of analysis a model can be used for. Each model can be used for `Session.AnalysisPeakPicking()` or `Session.AnalysisMcr()`. A model with non-empty `IntegrationRangeNames` is suitable for `Session.AnalysisIntegration()`, a model with non-empty `ComponentNames` is suitable for `Session.AnalysisComponentFitting()` or `Session.AnalysisHmfa()`, and a model with non-empty `CalibratedFeatureNames` is suitable for `Session.AnalysisPrediction()`.

Properties

Filename : string

Filename of the model.

Description : string

Description of the model as provided by the creator of the model.

IntegrationRangeNames : string[]

Names of integration ranges contained in the model.

ComponentNames : string[]

Names of hard model components contained in the model.

CalibratedFeatureNames : string[]

Names of calibrated features contained in the model.

2.1.8 PeakPickingResults Struct

The `PeakPickingResults` is a read-only structure returned by `Session.AnalysisPeakPicking()`.

Properties

DataSetUri : string

URI of the analyzed data set.

MinimumPeakHeight : double

Minimum height of peaks used for the analysis. This is either the user-specified input value to `AnalysisPeakPicking()` or an auto-detected value.

PeakPositionIndices : double[]

One-based X-indices of found peaks.

PeakPositions : double[]

X-values of found peaks.

PeakIntensities : double[]

Y-values of found peaks.

XData : double[]

X-values of the signal (after pre-treatments if any) used for the analysis.

YData : double[]

Y-values of the signal (after pre-treatments if any) used for the analysis.

2.1.9 IntegrationResults Struct

The `IntegrationResults` is a read-only structure returned by `Session.AnalysisIntegration()`.

Properties

DataSetUris : string[]

URIs of the analyzed data sets.

IntegrationRangeNames : string[]

Names of integration ranges used for the analysis.

IntegrationRangeIndices : int[]

One-based indices of integration ranges used for the analysis.

Areas : double[,]

2D array of calculated peak areas. Rows correspond to data sets; columns correspond to integration ranges.

2.1.10 ComponentFittingResults Struct

The `ComponentFittingResults` is a read-only structure returned by `Session.AnalysisComponentFitting()`.

Properties

DataSetUris : string[]

URIs of the analyzed data sets.

ComponentNames : string[]

Names of hard model components used for the analysis.

ComponentIndices : int[]

One-based indices of hard model components used for the analysis.

Weights : double[,]

2D array of calculated component weights. Rows correspond to data sets; columns correspond to hard model components.

2.1.11 PredictionResults Struct

The `PredictionResults` is a read-only structure returned by `Session.AnalysisPrediction()`.

Properties

DataSetUris : string[]

URIs of the analyzed data sets.

FeatureNames : string[]

Names of calibrated features used for the analysis.

FeatureIndices : int[]

One-based indices of calibrated features used for the analysis.

Values : double[,]

2D array of calculated feature values. Rows correspond to data sets; columns correspond to calibrated features.

RmsResiduals : double[,]

2D array of root mean square (RMS) spectral residuals. Rows correspond to data sets; columns correspond to calibrated features.

The array has identical columns for features predicted by the same Hard Model. The array contains NaN elements for features predicted by Integration Models.

RmsResidualsOutlierPValue : double[,]

2D array of probability values (p-values) for each spectral residual being an outlier. Rows correspond to data sets; columns correspond to calibrated features.

The array has identical columns for features predicted by the same Hard Model. The array contains NaN elements for features predicted by Integration Models.

MahalanobisDistance : double[,]

2D array of Mahalanobis distances of analyzed samples w.r.t. training samples. Rows correspond to data sets; columns correspond to calibrated features.

The array contains NaN elements for features not predicted by PLS Models.

MahalanobisDistanceOutlierPValue : double[,]

2D array of probability values (p-values) for each Mahalanobis distance being an outlier. Rows correspond to data sets; columns correspond to calibrated features.

The array contains NaN elements for features not predicted by PLS Models.

2.1.12 McrResults Struct

The `McrResults` is a read-only structure returned by `Session.AnalysisMcr()`.

Properties

DataSetUris : string[]

URIs of the analyzed data sets.

ComponentNames : string[]

Automatically generated names for each identified component.

S : double[,]

Spectral intensities (y-values) of identified components. Rows correspond to spectral x-values; columns correspond to components.

C : double[,]

Concentrations of identified components. Rows correspond to data sets; columns correspond to components.

RmsResiduals : double[]

Root mean square (RMS) spectral residuals for each data set.

Residuals are differences between measured and reconstructed signal.

R2 : double

Fraction of variance of the measured signal explained by the reconstructed signal.

XData : double[]

X-values of the signal (after pre-treatments if any) used for the analysis.

2.1.13 HmfaResults Struct

The `HmfaResults` is a read-only structure returned by `Session.AnalysisHmfa()`.

Properties

DataSetUris : string[]

URIs of the analyzed data sets.

ComponentNames : string[]

Names of pre-defined components (if any) and automatically generated names for each identified component (if any).

S : double[,]

Spectral intensities (y-values) of identified components. Rows correspond to spectral x-values; columns correspond to components.

C : double[,]

Concentrations of pre-defined and identified components. Rows correspond to data sets; columns correspond to components.

RmsResiduals : double[]

Root mean square (RMS) spectral residuals for each data set.
Residuals are differences between measured and reconstructed signal.

R2 : double

Fraction of variance of the measured signal explained by the reconstructed signal.

XData : double[]

X-values of the signal (after pre-treatments if any) used for the analysis.

2.2 COM API

The **COM API** is a set of classes contained in a COM-visible assembly you would use in scripts and applications that are COM-compliant, e.g. VBScript, Visual Basic, Excel, or LabVIEW. Before you can use it, you need to register the assembly using the Assembly Registration Tool (Regasm.exe) of the Microsoft .NET Framework 4.5. The assembly file is located at

<INSTALLPATH>\DLL\PeaxactApplicationServer.dll

Note: The PEAXACT Application Server requires Microsoft .NET Framework 4.5 or later to be installed on your computer.

Note: The assembly is registered automatically by the PEAXACT Installer.

ProgID

The COM API exposes the `IApplicationServer` class interface and the corresponding `ApplicationServer` class which is the only class you can create objects from. The `ApplicationServer` class can be referenced by its ProgID.

Version-specific ProgID: `PEXACT.ApplicationServer.4`

Version-independent ProgID: `PEXACT.ApplicationServer`

In Visual Basic, e.g., you would create a new instance of the `ApplicationServer` class with

```
Set pxAppServer4 = CreateObject("PEXACT.ApplicationServer.4")
Set pxAppServer = CreateObject("PEXACT.ApplicationServer")
```

Backward Compatibility

New versions of the COM API will be backward compatible. Therefore, it is highly recommended that you do not bind your application to a specific version of the Application Server DLL but instead bind dynamically to the newest version installed on the target computer. You can do this by referencing the version-independent ProgID.

Dynamic Binding vs. Static Binding

The Application Server DLL only allows for dynamic binding / late binding to assure that your application doesn't break when new methods or properties are added to the API in the future.

Exception Handling

The COM API throws exceptions with specific error messages. Use exception handling code appropriately for all method calls.

2.2.1 ApplicationServer Class

The `ApplicationServer` class is the root class of the API. It provides methods to `Start()` and `Stop()` the server. Once started you can create `Session` objects with the `NewSession()` or `LoadSession()` methods. A `Session` is a container for models and data sets, representing an isolated environment where analyses execute.

Constructor

The `ApplicationServer` class exposes a public constructor with no arguments. Although this would enable you to create multiple class instances, internally all instances share the same workspace. Therefore, it is recommended that you only create a single instance of the `ApplicationServer` class in your application. The ProgID of the `ApplicationServer` class is `PEXACT.ApplicationServer`.

Properties

IsStarted : bool

A flag indicating whether the Application Server has been started.

Methods

Start(string logFilename, string licenseFilename) : void

Starts the Application Server. The method does nothing if the server has been started already.

logFilename: Filename of a log file. (null or empty = no logging; "default" = default filename)

licenseFilename: Filename of a PEAXACT license file (extension LIC). (null or empty = detect automatically; "interactive" = show file dialog if required)

Stop() : void

Stops the Application Server. You should always call this method explicitly before exiting your application to properly clean up resources.

NewSession() : [Session](#)

Creates a new Session instance.

LoadSession(string filename) : [Session](#)

Creates a new Session instance by loading a PEAXACT session file.

filename: Filename of a PEAXACT session (extension PXS).

2.2.2 Session Class

A `Session` is a container for models and data sets, representing an isolated environment where analyses execute.

Constructor

The `Session` class has no public constructor. Instances of this class can only be created using the `NewSession()` or `LoadSession()` methods of the `ApplicationServer` class.

Properties

DataSetUris : string[]

Gets URIs of added data sets.

ModelFileNames : string[]

Gets filenames of added models.

IntegrationRangeNames : string[]

Gets names of integration ranges from all added models in the order models were added to the session.

ComponentNames : string[]

Gets names of non-empty hard model components from all added models in the order models were added to the session.

CalibratedFeatureNames : string[]

Gets names of calibrated features from all added models in the order models were added to the session.

Methods**Dispose() : void**

Frees the native resources associated with this object.

AddDataSet(string uri, bool removeExisting, object x, object y, params object[] z) : void

Add a data set to the session.

uri: URI of a data set to be added to the session.

removeExisting: Flag indicating whether to remove all other data sets from the session before adding the new one. This is useful for single-shot analyses.

x: Numeric array of x-values to be used instead of values read from the data file.

y: Numeric array of y-values to be used instead of values read from the data file.

z: Optional parameter/value pairs representing additional data set features. The parameter must be a string; the value must be of type double.

AddModel(string filename) : void

Add a model to the session.

filename: Filename of a PEAXACT model (extension PXM) to be added to the session. It can also be the name of a PEAXACT session file (extension PXS) containing any number of models to be added.

RemoveDataSet(object dataSetIdentifier) : void

Removes a data set from the session.

dataSetIdentifier: The one-based index (integer) or the URI (string) of the data set to be removed from the session.

RemoveModel(object modelIdentifier) : void

Removes a model from the session.

modelIdentifier: The one-based index (integer) or the filename (string) of the model to be removed from the session.

GetModelInfo(object modelIdentifier) : [ModelInfo](#)

Gets information about a model.

modelIdentifier: The one-based index (integer) or the filename (string) of the model in the session.

AnalysisPeakPicking(object dataSetId, object minPeakHeight) : [PeakPickingResults](#)

Performs peak picking for a single data set using pretreatments of the first added model (if any).

dataSetId: The one-based index (integer) or the URI (string) of a data set in the session.

minPeakHeight: Minimum height for peak detection. (null or empty = detect automatically)

AnalysisIntegration(object integrationRangeIdentifier) : [IntegrationResults](#)

Performs peak integration for all added data sets using all added integration models.

integrationRangeIdentifier: One-based indices (integer array) or names (string array) of integration ranges (as in `IntegrationRangeNames`) to be considered for the analysis.

AnalysisComponentFitting(object componentIdentifier) : [ComponentFittingResults](#)

Performs component fitting for all added data sets using all added hard models.

componentIdentifier: One-based indices (integer array) or names (string array) of hard model components (as in `ComponentNames`) to be considered for the analysis.

AnalysisPrediction(object featureIdentifier) : [PredictionResults](#)

Performs feature prediction for all added data sets using all added calibration models.

featureIdentifier: One-based indices (integer array) or names (string array) of calibrated features (as in `CalibratedFeatureNames`) to be considered for the analysis.

AnalysisMcr(int numComponents, params object[] options) : [McrResults](#)

Performs MCR-ALS for all added data sets using pretreatments of the first added model (if any).

numComponents: Number of unknown components to be identified from the data.

options: Additional options for the algorithm as parameter/value-pairs. The parameter can be any of the following strings: `C0`, `ToleranceRmse`, `MaxIterations`, `MaxUnsuccessfulAttempts`, `IsNonnegativeC`, `IsNonnegativeS`, `IsUnimodalC`, `IsClosureC`. The meaning of these parameters and appropriate values are identical to the .NET API as documented in 2.1.5.

AnalysisHmfa(int numComponents, params object[] options) : [HmfaResults](#)

Performs HMFA for all added data sets using the first added model.

numComponents: Number of unknown components to be identified from the data.

options: Additional options for the algorithm as parameter/value-pairs. The parameter can be any of the following strings: `IsClosureC`. The meaning of these parameters and appropriate values are identical to the .NET API as documented in 2.1.6.

2.2.3 ModelInfo Struct

See Section 2.1.7

2.2.4 PeakPickingResults Struct

See Section 2.1.8

2.2.5 IntegrationResults Struct

See Section 2.1.9

2.2.6 ComponentFittingResults Struct

See Section 2.1.10

2.2.7 PredictionResults Struct

See Section 2.1.11

2.2.8 McrResults Struct

See Section 2.1.12

2.2.9 HmfaResults Struct

See Section 2.1.13

2.3 Programming Examples

2.3.1 Using the .NET API in C#

This example demonstrates how to use the .NET API in C#. The program uses a calibrated model to predict features from a measured sample.

Note: You have to reference the `PeaxactApplicationServer.dll` assembly first.

```
using System;
using S_PACT.PEAXACT;

namespace Examples
{
    class PredictionExample
    {
        Session pxSession;

        static void Main()
        {
            PredictionExample example = new PredictionExample();
            example.Run();
            Console.WriteLine("Press any key to continue.");
        }
    }
}
```

```

        Console.ReadKey();
    }

    private void Run()
    {
        // Start the Application Server
        StartOptions startOptions = new StartOptions() {
            LicenseFilename = @"c:\license.lic",
            LogFilename = @"c:\appserver.log" };
        ApplicationServer.Instance.Start(startOptions);
        // Create a new session
        pxSession = ApplicationServer.Instance.NewSession();
        // Add model and data, then run the analysis
        pxSession.AddModel(@"c:\user\model.pxm");
        LoadData();
        PredictFeatures();
        // Stop the Application Server (clean up unmanaged resources)
        ApplicationServer.Instance.Stop();
    }

    private void LoadData()
    {
        // This function demonstrates how to add data sets with xy-data
        int nx = 800; // number of data points
        double[] xData = new double[nx]; // vector of x-data, e.g. wavenumbers
        double[] yData = new double[nx]; // vector of y-data, e.g. intensities

        // populate xData and yData
        // ...

        // Add data set. The URI can be arbitrary if xy-data is provided.
        pxSession.AddDataSet("dummy.xyz", true, xData, yData);
    }

    private void PredictFeatures()
    {
        // Run the analysis for all calibrated features
        PredictionResults r = pxSession.AnalysisPrediction();

        // Display results
        for (int j = 0; j < r.FeatureNames.Length; j++)
        {
            Console.WriteLine("{0} = {1}", r.FeatureNames[j], r.Values[0, j]);
        }
    }
}
}

```

2.3.2 Using the COM API in VB Script

This example demonstrates how to call the PEAXACT Application Server from Visual Basic Script (VBS). VBS only supports the COM API. The script uses a model for integration of peak areas of two chromatograms and displays results.

Note: If you installed the 32-bit version of the Application Server on a 64-bit OS, you would need to run the script with a 32-bit version of `wscript.exe`, which can be found at `%windir%\SysWOW64\wscript.exe`

```

' Create instance of ApplicationServer
Set pxAppServer = CreateObject("PEAXACT.ApplicationServer")

```

```
' Start the server with a default log file and interactive license activation
pxAppServer.Start "default", "interactive"
Set pxSession = pxAppServer.NewSession

' Add one model and multiple data sets
pxSession.AddModel "c:\model\file.pxm"
pxSession.AddDataSet "c:\data\chromatogram.csv#1", false, null, null
pxSession.AddDataSet "c:\data\chromatogram.csv#2", false, null, null

' Integrate peak areas of "Range 1" according to model specifications
Set results = pxSession.AnalysisIntegration("Range 1")

' Display results
For i = 0 To UBound(results.DataSetUris)
    ' Note: Indexing into class property arrays is not directly possible in VBS
    uris = results.DataSetUris
    areas = results.Areas
    Wscript.Echo "Peak area (Range 1) for sample " & uris(i) & " = " areas(i, 0)
Next

' Stop the server
pxAppServer.Stop
```

2.3.3 Dynamically bind to future versions of the Application Server assembly (.NET API)

This example demonstrates how you could make your .NET application independent of a specific version of the Application Server and instead bind to the newest Application Server assembly installed on the target computer. This is possible because new versions of the .NET API will be backward compatible.

Note: It is highly recommended NOT to bind to a specific version of the Application Server assembly because you would need to upgrade your application each time a new PEAXACT version gets released.

```
using Microsoft.Win32;
using S_PACT.PEAXACT;
using System;
using System.IO;
using System.Reflection;

namespace Examples
{
    class DynamicBindingExample
    {
        static DynamicBindingExample()
        {
            // Add assembly resolver within a static constructor
            AppDomain.CurrentDomain.AssemblyResolve +=
                new ResolveEventHandler(MyResolveEventHandler);
        }

        static void Main(string[] args)
        {
            // Start the Application Server, resolve assembly if necessary.
            ApplicationServer.Instance.Start();
        }
    }
}
```

```

private static Assembly MyResolveEventHandler(object s, ResolveEventArgs e)
{
    // Return if assembly has been loaded already
    Assembly[] loadedAssemblies = AppDomain.CurrentDomain.GetAssemblies();
    foreach (Assembly assembly in loadedAssemblies)
    {
        if (assembly.FullName.StartsWith(e.Name))
            return assembly;
    }

    if (e.Name.StartsWith("PeaxactApplicationServer"))
    {
        // Search registry for the highest installed version
        RegistryKey key, subKey;
        try
        {
            key = Registry.LocalMachine.OpenSubKey(@"SOFTWARE\S-PACT");
            if (key == null) return null;
        }
        catch
        {
            return null;
        }

        UInt32 version = 0, maxVersion = 0;
        String installPath = "", assemblyPath = "", validAssemblyPath = "";
        foreach (String keyName in key.GetSubKeyNames())
        {
            if (!keyName.StartsWith("PEAXACT Application Server"))
                continue;
            try
            {
                {
                    version = UInt32.Parse(keyName.Substring(26));
                    if (version <= maxVersion) continue;
                    maxVersion = version;
                    subKey = key.OpenSubKey(keyName);
                    if (subKey == null) continue;
                    installPath= subKey.GetValue("InstallPath", "").ToString();
                    if (installPath == "") continue;
                    assemblyPath = Path.Combine(installPath,
                        "PeaxactApplicationServer.dll");
                    if (File.Exists(assemblyPath))
                        validAssemblyPath = assemblyPath;
                }
            }
            catch { }
        }

        // Load assembly from installation path
        if (validAssemblyPath != "")
            return Assembly.LoadFrom(validAssemblyPath);
    }
    return null;
}
}
}

```

3 CUSTOM INTERFACES

3.1 OPUS Process

Note: A video on how to integrate the PEAXACT Application Server with OPUS Process is published on the [PEAXACT Blog](#).

3.1.1 Prerequisites

Software Requirements

- OPUS 6.5 or higher
- PEAXACT 4 or higher. A 32-bit installation is required!

OPUS 7 Workaround

The following workaround is necessary for OPUS version 7 to work with PEAXACT:

- 1) Open the Windows Explorer and open the OPUS installation directory
- 2) Rename file `Calo.dll` to `Calo.dll_hidden` or any other name, such that the file won't be found by OPUS any more

Please note that this workaround disables OPUS support for Unscrambler.

Additional Files

These instructions refer to a special OPUS script file named `PEAXACTComponentAnalysis.obs`. The file is used as a placeholder during set-up of an OPUS PROCESS scenario and does nothing so far. The file is located at `<INSTALLPATH>\DLL\OPUS`.

3.1.2 OPUS Configuration

- 1) Run the [diagnosis program](#) first to test whether the PEAXACT Application Server is installed and registered correctly.
- 2) Configure a new OPUS PROCESS scenario file (.obs) with the OPUS scenario browser
 - a) Each measurement point requires a "No Evaluation" data channel for triggering the measurement (must be the first data channel in each case).
 - b) Add data channels with data evaluation by script `PEAXACTComponentAnalysis.obs`
- 3) Modify the scenario script according to instructions in the next Section
- 4) Run the process script in OPUS.

3.1.3 Modifying OPUS scenario file

Important Notes

- Set-up the whole OPUS PROCESS scenario first using the OPUS scenario browser.
- Run and test the scenario before making manual modifications to the scenario file.
- Once the scenario script is modified manually, the scenario should not be changed with the OPUS scenario browser anymore because this would overwrite all manual modifications. Again, make sure to finish all steps in the scenario browser first.
- Use the OPUS script editor (Menu File > Open > *.obs) to modify the scenario script as follows below. If you copy and paste text from a PDF version of this document, copy each page separately because this will preserve line breaks and also prevents from copying headers and footers.

At the beginning of the script, after `Option Explicit` add:

```
' Added by S-PACT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Dim pxAppServer, pxSession
' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

At the beginning of sub-procedure `Form_OnLoad()` add:

```
' Added by S-PACT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MsgBox "Starting PEAXACT Application Server..."
Dim logFile, licenseFile
Set pxAppServer = CreateObject("PEAXACT.ApplicationServer")
logFile = ""
licenseFile = "interactive"
pxAppServer.Stop
pxAppServer.Start logFile, licenseFile
Set pxSession = pxAppServer.NewSession
pxSession.AddModel "<modelFilename>"
' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Customize `<modelFilename>` to load your models

- Substitute `<modelFilename>` with the full path and filename of a PEAXACT model. For instance, the line would then read:


```
pxSession.AddModel "C:\models\cyclohexaneModel.pxm"
```
- If you want to add more models, duplicate the `pxSession.AddModel` line.

At the beginning of sub-procedure `Form_OnUnload()` add:

```
' Added by S-PACT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pxAppServer.Stop
' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

At the very end of the script, add:

```
' Added by S-PACT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Function PEAXACTAnalysis(ByVal typeOfAnalysis, ByVal block, ByVal Id)
    Dim vntResult, iPoint, nPoints, firstX, lastX, path, file, URI
    Dim xData(), yData(), resultObj, result
    ' set some options and read data
    vntResult = Form.OpusRequest("BINARY")
    vntResult = Form.OpusRequest("FLOAT_MODE")
    vntResult = Form.OpusRequest("FLOATCONV_MODE ON")
    vntResult = Form.OpusRequest("DATA_POINTS")
    vntResult = Form.OpusRequest("READ_FROM_BLOCK " & block)
    vntResult = Form.OpusRequest("READ_PARAMETER PAT")
    path = split(vntResult, chr(10))(1)
    vntResult = Form.OpusRequest("READ_PARAMETER NAM")
    file = split(vntResult, chr(10))(1)
    vntResult = Form.OpusRequest("READ_PARAMETER NPT")
    nPoints = split(vntResult, chr(10))(1)
    vntResult = Form.OpusRequest("READ_PARAMETER FXV")
    firstX = split(vntResult, chr(10))(1)
    vntResult = Form.OpusRequest("READ_PARAMETER LXV")
    lastX = split(vntResult, chr(10))(1)
    ' create xData
    ReDim xData(nPoints-1)
    For iPoint = 0 To nPoints-1
        xData(iPoint) = Cdbl(firstX + iPoint * (lastX-firstX)/(nPoints-1))
    Next
    ' read yData
    vntResult = Form.OpusRequestData("READ_DATA", yData)
    For iPoint = 0 To nPoints-1
        yData(iPoint) = Cdbl(yData(iPoint+1))
    Next
    ReDim Preserve yData(nPoints-1)
    ' add data set and perform the analysis
    URI = path & chr(92) & file & "#" & block & "-1"
    pxSession.AddDataSet URI, true, xData, yData
    Select Case UCase(typeOfAnalysis)
        Case "INTEGRATION"
            Set resultObj = pxSession.AnalysisIntegration(Id)
            result = resultObj.Areas
        Case "COMPONENTFITTING"
            Set resultObj = pxSession.analysisComponentFitting(Id)
            result = resultObj.Weights
        Case "PREDICTION"
            Set resultObj = pxSession.analysisPrediction(Id)
            result = resultObj.Values
        Case "PREDICTIONOUTLIERPLS"
            Set resultObj = pxSession.analysisPrediction(Id)
            result = resultObj.MahalanobisDistanceOutlierPValue
        Case "PREDICTIONOUTLIERIHM"
            Set resultObj = pxSession.analysisPrediction(Id)
            result = resultObj.RmsResidualsOutlierPValue
        Case Else : MsgBox "Invalid typeOfAnalysis: " & typeOfAnalysis
    End Select
    PEAXACTAnalysis = vbLf & vbLf & CStr(result(0,0)) ' set output
End Function
' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Search and replace the placeholder script

- Press CTRL+F3 to open the text search dialog
- Search for `PEAXACTComponentAnalysis.obs` (ignore any matches found in the first line)
- A matching line should start with `vntResult = Form.OpusRequest("VBScript`

- Replace the whole line by

```
' Modified by S-PACT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
vntResult = PEAXACTAnalysis("<typeOfAnalysis>", "<block>", "<componentName>")
' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- Substitute `<typeOfAnalysis>` with one of the following types:
 - `integration` – calculation of peak area; requires an integration model
 - `componentFitting` – calculation of component weight; requires a hard model
 - `prediction` – prediction of feature value; requires a calibration model
 - `predictionOutlierPLS` – calculates the probability (p-value) for a spectral outlier towards a PLS model; requires a PLS calibration
 - `predictionOutlierIHM` – calculates the probability (p-value) for a spectral outlier towards an IHM model; requires an IHM calibration
- Substitute `<block>` with the desired file block, e.g. `AB`
- Substitute `<componentName>` depending on your choice of `<typeOfAnalysis>`:
 - `integration`: substitute with the name of an integration range
 - `componentFitting`: substitute with the name of a component model
 - `prediction`, `predictionOutlierPLS`, `predictionOutlierIHM`: substitute with the name of a calibrated feature. Be careful not to accidentally use names of integration ranges or component models. Calibrated feature names can be found in the model summary report:

CALIBRATION SUMMARY	
Calibration method	IHM ratiometric
Features	3
Feature names	Cyclohexane, Dioxane, Toluene
Linked component models	Cyclohexane, Dioxane, Toluene
Training samples	8

- For instance, the line would now read:

```
vntResult = PEAXACTAnalysis("prediction", "AB", "Cyclohexane")
```

Note: `<componentName>` can also be the component's index. The index is consecutively numbered across all added models. For instance, the line would read:

```
vntResult = PEAXACTAnalysis("prediction", "AB", 1)
```

Do not enclose the index in double quotes! Use the index instead of the name when multiple components have identical names.

- Repeat this step until all occurrences of `PEAXACTComponentAnalysis.obs` are replaced.

3.2 HoloPro

3.2.1 Prerequisites

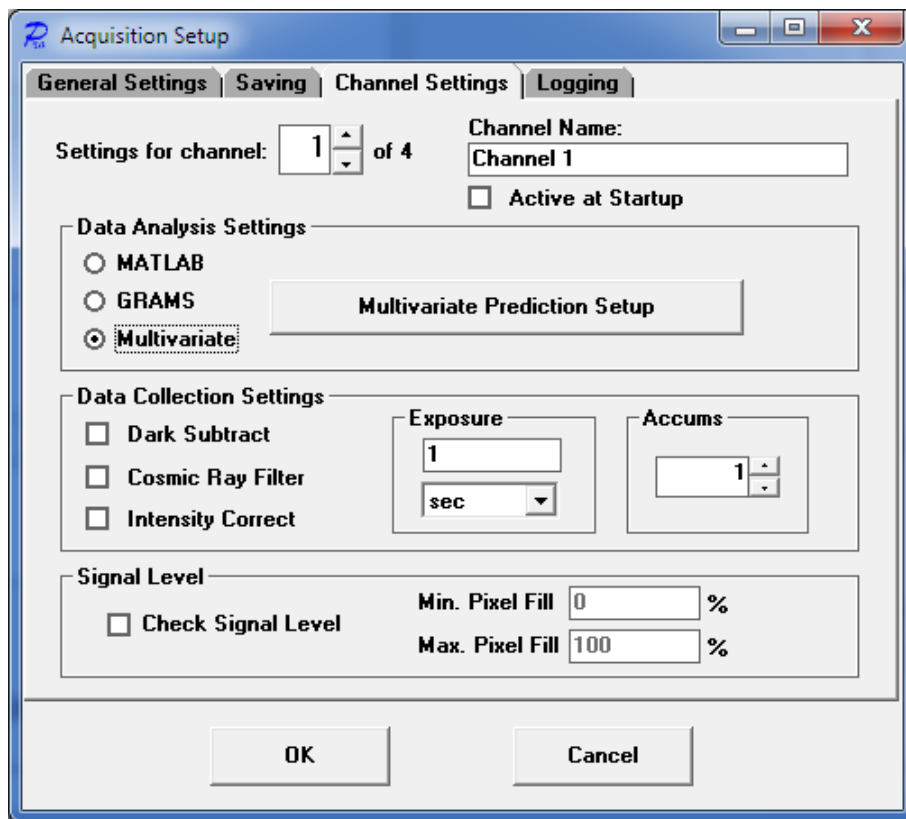
Software Requirements

- HoloPro 3.2.0.6 or higher (expected to be installed in directory `C:\HoloPro`)

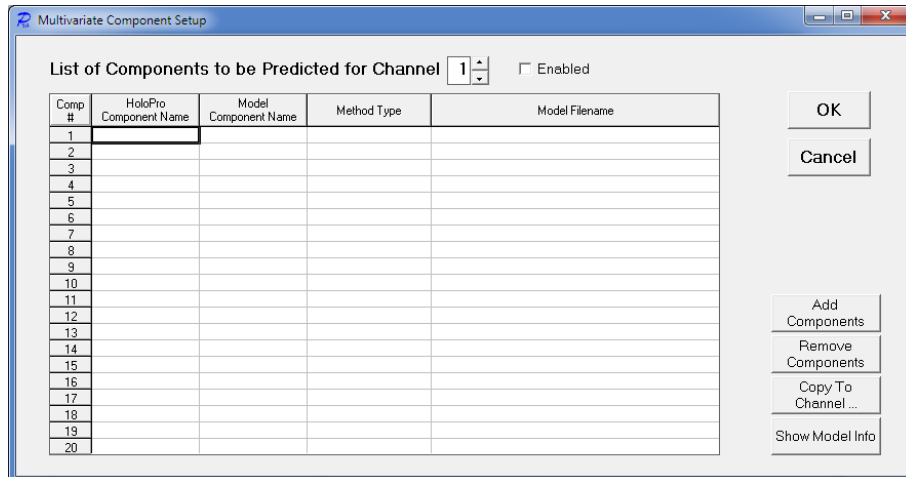
- PEAXACT 4 or higher. A 32-bit installation is required.

3.2.2 Configuration

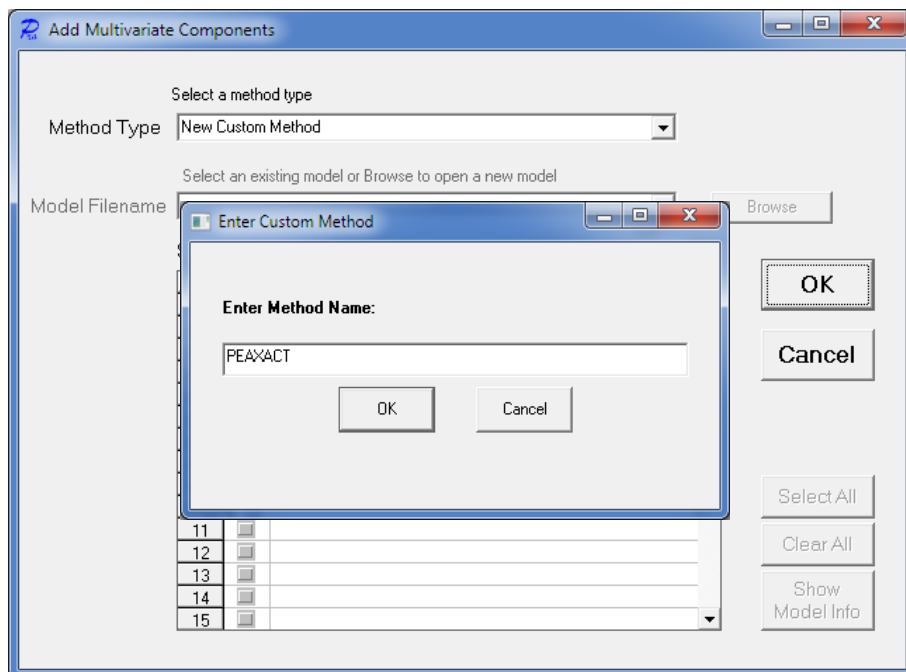
- 1) Run the [diagnosis program](#) first to test whether the PEAXACT Application Server is installed and registered correctly.
- 2) Start HoloPro and open the **Channel Settings** (menu **Settings > Acquisition Setup**)



- 3) Tick **Multivariate** in the Data Analysis Settings Panel, then click the **Multivariate Prediction Setup** button
- 4) At the top of the In the next window, select a channel, then click the **Add Components** button

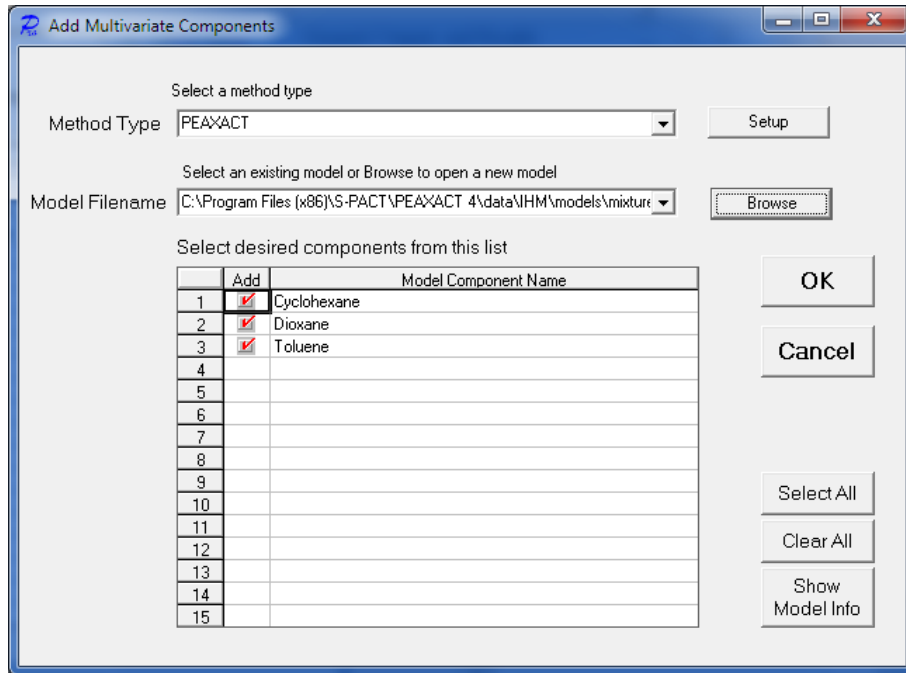


- 5) From the **Method Type** list select **New Custom Method** and enter PEAXACT (or select PEAXACT if it has already been added before).



Note: Optionally, after step 5, you may click the **Setup** button which appears next to the Method Type in order to change PEAXACT logging options. In case of unexpected errors you should enable logging and read the log file. Otherwise you should keep logging disabled!

- 6) **Browse** for a calibrated model file and select components. Close with **OK**.



Note: Adding the first model may take a while (because the PEAXACT Application Server gets started in the background).

- 7) You can add more components from other models to the same channel, or you can add components to other channels by repeating steps 4 to 6.

Note: In step 6, you can also browse for a PEAXACT session file in order to load multiple models from the session.

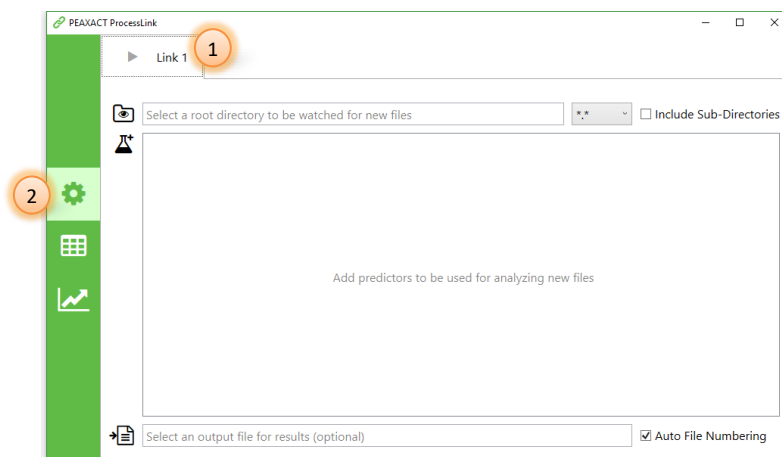
- 8) After closing all setup windows with **OK** you may start measuring. The prediction of feature values takes place after each measurement. Results will be displayed in the main window of HoloPro.


4 PEAXACT PROCESSLINK

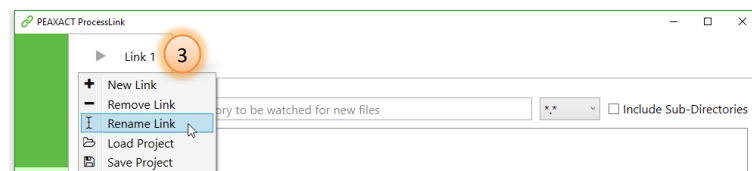
To start the PEAXACT ProcessLink App, select **PEAXACT ProcessLink** from the Windows start menu. The application window by default contains a new Link, awaiting user [setup](#).

Note: PEAXACT ProcessLink requires a license valid for module "ProcessLink"

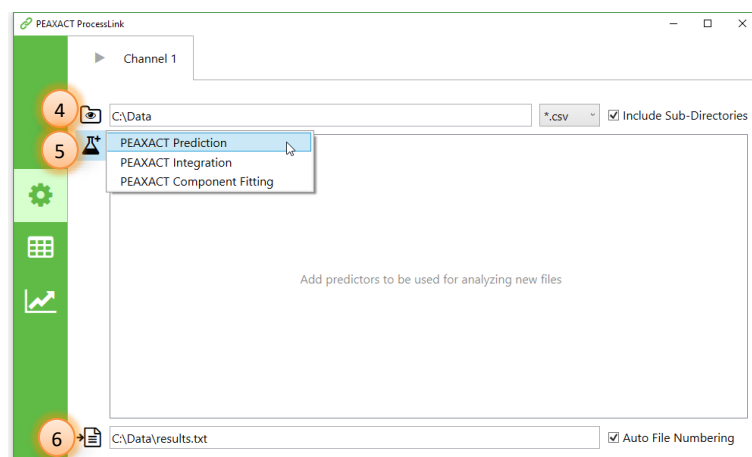
4.1 Setup






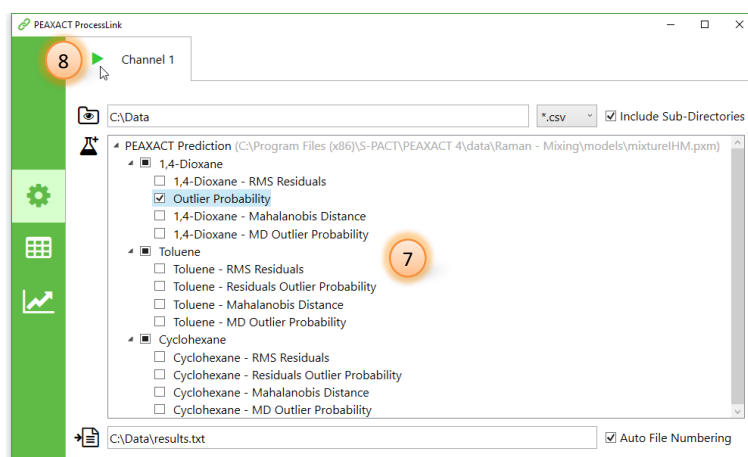
- (1) Select the Link you want to configure.
- (2) Click  to select the setup view.





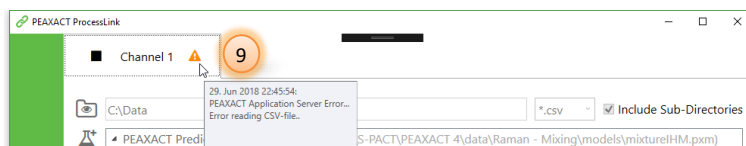
- (3) To rename the link, double-click the Link name or right-click the tab and select **Rename Link** from the context menu. The name will be used as a node name on the OPC UA server and as title for the real-time chart.



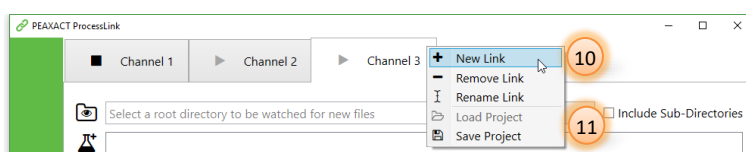
- (4) Select a directory which should be monitored for incoming files. Click  to browse for a folder or manually type in the path. Optionally, choose a pre-defined file filter or enter a custom file filter, and choose whether sub-directories should be monitored as well.
- (5) Add predictors to be used for analyzing new files. First click , then select a predictor from the list, and finally use the file browser to select one or more PEAXACT model files.
- (6) If results should be written to a file, click  to browse for a file or manually type in the filename. Optionally, choose whether duplicate files should be numbered automatically.



- (7) The list of available results provided by each predictor is displayed in the central tree view. Tick / untick the results you want to include / exclude. To remove a predictor from the list, select the top-level item and press the [Del] key. To rename any result, select it and press the [F2] key. Sub-results will be renamed automatically if they still have the default name, but they can be given individual names as well. The result names are used as headers in the result file, as display names in the real-time table and real-time chart, and as node names on the OPC UA server.
- (8) Click  to start the folder monitoring. This will also create the result file (if specified) and create nodes on the OPC UA server. Once started, you cannot change the configuration anymore until you click  to stop the folder monitoring.



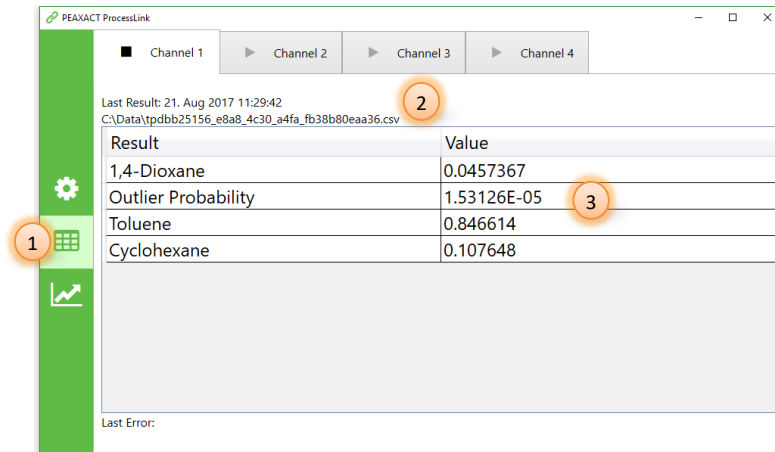
- (9) In case of errors, hover the mouse over the notification icon  to see details.




- (10) You can add, configure, and start up to 5 concurrent Links.

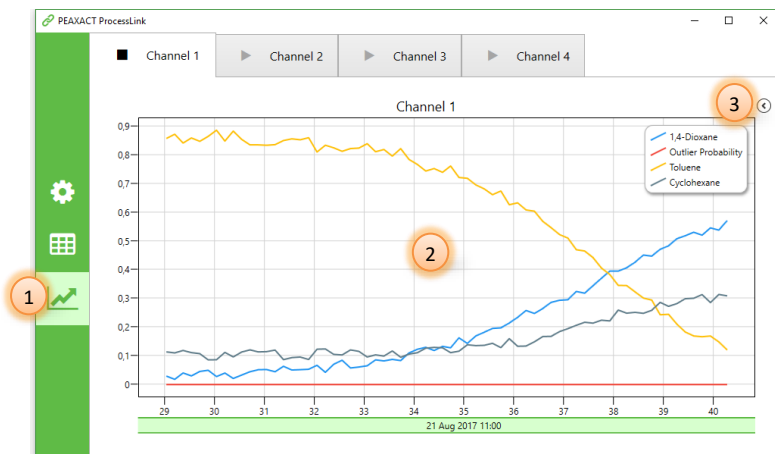
- (11) The whole setup of all Links can be saved to a project file. You can load a project file later in order to re-use a previously saved setup. Note that loading a project file will replace the current setup. Also, a project can only be loaded if no Links are started.



4.2 Real-time Table



- (12) Click  to switch to the real-time table view.
 (13) The last file which was analyzed successfully is displayed at the top.
 (14) The last result values are displayed in the table.

4.3 Real-time Chart



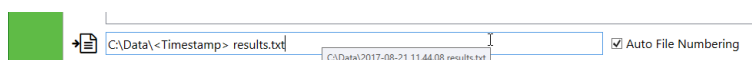
- (1) Click  to switch to the real-time chart view.
 (2) The last 1000 result values are plotted vs. time.
 (3) Click  to expand the side menu.



- (4) Tick / untick the line graphs you want to show / hide.
- (5) Right-click on the chart to open the context menu with additional options.

4.4 Result File

If a result filename is specified during setup, values will be appended to that file whenever new results are available. When specifying the filename, you can use various placeholders (e.g. <Timestamp>) which will be replaced with actual text when the Link gets started. Hover the mouse over the filename field to get a filename preview (if the Link is not started) or to get the actual used filename (after the Link is started).



The following placeholders are valid (case-sensitive):

<Timestamp>	Current date and time, shortcut for <Date> <Time>
<Date>	Current date, shortcut for <yyyy>-<MM>-<dd>
<Time>	Current time, shortcut for <HH>.<mm>.<ss>
<d>	The day of the month, from 1 through 31.
<dd>	The day of the month, from 01 through 31.
<ddd>	The abbreviated name of the day of the week.
<dddd>	The full name of the day of the week.
<h>	The hour, using a 12-hour clock from 1 to 12.
<hh>	The hour, using a 12-hour clock from 01 to 12.
<H>	The hour, using a 24-hour clock from 0 to 23.
<HH>	The hour, using a 24-hour clock from 00 to 23.
<m>	The minute, from 0 through 59.
<mm>	The minute, from 00 through 59.
<M>	The month, from 1 through 12.
<MM>	The month, from 01 through 12.
<MMM>	The abbreviated name of the month.

<MMMM>	The full name of the month.
<s>	The second, from 0 through 59.
<ss>	The second, from 00 through 59.
<t>	The first character of the AM/PM designator.
<tt>	The AM/PM designator.
<y>	The year, from 0 to 99.
<yy>	The year, from 00 to 99.
<yyy>	The year, with a minimum of three digits.
<yyyy>	The year as a four-digit number.

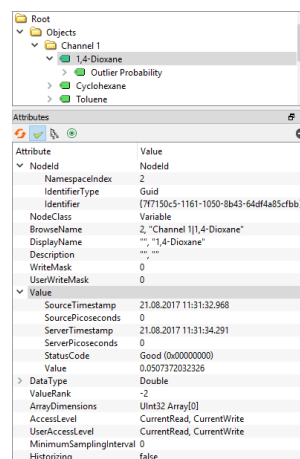
4.5 OPC UA Server

PEAXACT ProcessLink contains a built-in OPC UA server which gets started automatically when the first Link gets started. The server then stays alive until the ProcessLink application shuts down.

The server configuration is as follows:

Server Name	PEAXACT ProcessLink
Endpoint URL	opc.tcp://localhost:36090
Supported Security Policies	None, Basic256Sha256
Supported Message Security Modes	None, Sign&Encrypt
Supported User Identity Token Types	Anonymous

The server contains a top-level node for each started Link. The top-level node is named after the Link and contains data nodes which represent the results. E.g., the following figure demonstrates the OPC data structure and shows some attributes of the data nodes.



The `StatusCode` attribute of the nodes will be `BadWaitingForInitialData` before the first value is available, `BadOutOfService` after a Link gets stopped, and `Good` otherwise.

4.6 OPC Security Setup

For testing purposes and for applications which don't require special security considerations, the OPC UA server supports the security policy `None`. This allows for a quick and easy client-server-connection as no additional configuration is required.

When PEAXACT ProcessLink is used in a production environment, clients should connect to the server endpoint using the `Basic256Sha256` security policy and the `Sign&Encrypt` message security mode to assure two-way secure communication between both parties. Additional configuration steps are required to exchange certificates between server and client. The procedure is partly automated but also involves the manual copying of certificate files on the server computer:

- 1) Connect the client to the secure server endpoint. The server will send its certificate back to the client. Most clients provide some dialog to install and accept the certificate, in which case you should do so and continue with step 2. If the client does not provide such a dialog you have to manually copy and install the server certificate file to the client computer. Please refer to the client's user manual. You can find the server's certificate file in the following directory on the server computer:

```
%ProgramData%\S-PACT\PEAXACT ProcessLink\pki\own\certs
```

- 2) In step 1, the client should have also sent its certificate to the server. The server does not provide a dialog to automatically accept the client certificate. Instead, the server stores certificates of newly connecting clients at

```
%ProgramData%\S-PACT\PEAXACT ProcessLink\pki\rejected\certs
```

from where you have to move the file to the following directory:

```
%ProgramData%\S-PACT\PEAXACT ProcessLink\pki\trusted\certs
```

Only if both parties have installed and trusted each other's certificate the client will be able to connect to the server. The server will then encrypt each message so that only the client is able to read it. The server will also digitally sign each message so that the client can check whether it was actually sent by the server and has not been tampered with. Attackers who happen to intercept a message won't be able to read or modify it.

4.7 Command Line Usage

PEAXACT ProcessLink can be started with additional parameters from the command line, from a script or batch file, or on Windows startup.

Parameters in square brackets are optional. Angle brackets represent placeholders which must be replaced by specific values.

```
PeaxactProcessLink.exe [-project <file> [-activate]]
```

```
-project <file> loads a project file
```

```
-activate starts all Links that can be started. To be used in combination  
with -project
```

4.8 Additional Notes

- If an error occurs during the analysis of a new file, no changes are made to the result text file, to the OPC UA server, or to the real-time table and chart.
- A started Link will stop silently if the monitored directory gets deleted.
- A started Link will continue working if the monitored directory gets renamed.
- New files written to the monitored directory will be ignored if the analysis of the previous file hasn't finished yet.

5 TROUBLE SHOOTING

Problems with the COM API

Symptoms

You cannot access the PEAXACT COM API from your third-party application.

Resolution

In case of any problems with the PEAXACT COM API you should try the following

- 1) Run the [diagnosis program](#)

After starting the program it performs several tests. In case of errors a possible solution is suggested. You have to fix all problems before you can use the interface correctly.

- 2) Under some circumstances the diagnosis program crashes (throwing a Windows error) when the COM DLL is registered incorrectly. If this happens, you have to register the DLL manually by executing the file

```
<INSTALLPATH>\DLL\COM\register.bat
```

Note that administrator privileges are required to execute the file. Afterwards, run the diagnosis again.

Problems with the HoloPro Custom Interface

Symptoms

You receive an error when trying to add PEAXACT as new Custom Method in HoloPro.

Resolution

Make sure you have installed the 32-bit version of PEAXACT even on Windows 64-bit. Then see [Problems with the COM API](#)