SYSPRO Open Reporting API

SYSPRO 8

Reference Guide

Published: August 2023

SYSPRO

CONTENTS

Open Reporting API

Open Reporting API

Exploring

Where it fits in?

The **Open Reporting API** lets developers and external applications call SYSPRO to run and distribute documents directly in the external application.

Leveraging the SYSPRO Reporting Service Server infrastructure, developers can query the SYSPRO database and produce the required documents which are added to the print queue from where they can be viewed, executed and managed. Developers can access the document via the document queue for further automation (a destination code indicates the origin of the queue item).

Functionality

The **Open Reporting API** enables custom application or third party developers to programmatically produce single documents from applications outside the main SYSPRO application, by referencing an assembly located within the SYSPRO `\Base` folder.

The **Open Reporting API** works by creating a business object wrapper around the standard SYSPRO print programs and by providing business objects to retrieve the required information about those documents that are printed. Please refer to the **Referencing** section for more information on the XML that is used for each document type.

Documents that have already been generated and stored in the document queue can't be retrieved via the API.

Each document type first needs to be converted to support the **Open Reporting API** before the wrapper business objects can be written to call them. This is to ensure that the SYSPRO print programs don't try to show any user interface when called by the API. As a result, not all document types may be supported by the API.

Disclaimer:

The business objects used by the **Open Reporting API** are intended for use within the API only and should not be used directly in e.net. SYSPRO reserves the right to change the way these business objects work in order to support the API.

Starting

Prerequisites

Before using the **Open Reporting API**, ensure that **Server-side Printing** is configured correctly in SYSPRO.

The following is required:

Server Technology Requirements

- Reporting Host Service
- Crystal Reports Server Embedded (CR 2016 and any specific service packs)
- SAP BusinessObjects BI platform .Net SDK Redistributable 64-bit 4.2. SP3

Client Configuration Requirements

- Browser pop-ups must be enabled

> This is only applicable for previewing PDFs in a web-browser application.

SYSPRO Configuration Requirements

- Server-side reporting (configured within the SYSPRO **Setup Options** program *Setup Options > System Setup > Reporting*).
- e.Net Service Details (configured within the SYSPRO **Setup Options** program *Setup Options > System Setup > Artificial Intelligence*).
- SMTP emailing (configured within the SYSPRO **Setup Options** program *Setup Options > System Setup > Connectivity*).

Restrictions and Limits

- The **Open Reporting API** is only supported for SRS server-side printing, as it makes use of the **SYSPRO 8 Reporting Host Service** to call the print business objects and generate documents.

> The **Open Reporting API** is therefore not available for client-side printing.

- When emailing a document, only one email address can be entered in the **To** and **CC** lines.

- Known limitations:

  The following are known limitations, which may be addressed during the lifetime of ***SYSPRO 8***:

  - Batch printing of documents is currently not supported.
  - Report printing is not supported.
  - Documents that are printed using this architecture are currently not added to the document archive.

    The generated documents are stored in the SRS document queue used for server side printing and will be visible from the **SRS Document Queue** program. However, they are not added to the document archive.

    There will be no PDF file in the **SRS Document Printing** archive folder, nor will the document be visible in the **Document Archive Viewer**.
  - This functionality is limited to the following document types:
    - Sales order documents
      - Invoices
      - Delivery notes
      - Order acknowledgments
      - Dispatch notes
      - Dispatch note invoices
    - AR statements
    - Quotations
    - Factory documentation
    - Foreign and local purchase orders (excluding requisitions)

    

    Support for other documents and reports may be made available in the future.

Configuring

Setup Options

The **Setup Options** program lets you configure how SYSPRO behaves across all modules. These settings can affect processing within this program.

E.Net Service Details System Setup

Setup Options > System Setup > E.Net Service Details

- Server name
- SOAP port
- REST port

Reporting System Setup

Setup Options > System Setup > Reporting

- Reporting configuration
- Server-side configuration

Solving

FAQs

Where can I download the SYSPROSRSClientLibrary.dll?

The `SYSPROSRSClientLibrary.dll` assembly file is located in your SYSPRO `\Base` folder.

It is placed into this folder during the installation of ***SYSPRO 8***.

Where can I find the SYSPROSRSDocumentAPI.dll assembly?

This file is no longer in use and has been replaced by the `SYSPROSRSClientLibrary.dll` assembly file.

The `SYSPROSRSClientLibrary.dll` assembly (located in your `\Base` directory) is used by all applications that want to use the functionality of the SRS API.

All DLLs are located in your `\Base` directory.

Where can I see the host service end-point in SYSPRO?

1. Load the **System Setup** program (*SYSPRO Ribbon bar > Setup > General Setup*).
2. Navigate to the **Reporting** tab.
3. The host service end-point is displayed at the **REPORTING SERVICE** field of the **SERVER-SIDE CONFIGURATION** section.

Which assembly should be used for which functionality?

1. The `SYSPROSRSClientLibrary.dll` assembly is used by all applications that want to use the printing or reporting services functionality.
2. The `SysproWCFClientLibrary40.dll` assembly can be used for all applications that want to access SYSPRO.

The `SysproWCFClientLibrary40.dll` is not a requirement for printing.

Could the API be used for SWS to email the document?

Yes, you can produce a document from within SYSPRO workflow if you create a custom activity that will allow you to create a reference to the client library assembly and write your C# code.

Using

Process

Installing/starting the SRS Host Service

The **SYSPRO 8 Reporting Host Service** is deployed via the **SYSPRO Installer** and can be installed automatically when ***SYSPRO 8*** is installed.

We recommend stopping the **SYSPRO 8 Reporting Host Service** before any updates are done and restarting it thereafter.

You only need to reinstall the service if an updated and/or improved version of the service is released.

Configuring server-side reporting

1. From the **Setup Options** program, select the **Reporting System Setup** form (*Setup Options* > *System Setup* > *Reporting*).
2. Configure the required settings:

Field	Action
Reporting configuration	Select **Server-side reporting using SQL**.
SQL Server Name	Enter the name of the SQL Server instance that contains the _SRS database.
Reporting authentication	Select the type of authentication you will use.

3. Save your changes.

   Ensure that you can successfully print the document type using SYSPRO before trying to print it using the API.

How to use the API

All API calls are done via the SRS library.

1. Create a new project and add the `DLL` as a reference.
2. Use the five available methods to query and generate documents.

How to add a reference to the client library assembly

1. Open the solution.
2. Select the **Solution Explorer** (*View* > *Solution Explorer*).
3. Right-click on **References** in the **Solution Explorer** pane of the project and select **Add Reference**.
4. Select **Browse** and navigate to the SYSPRO `\Base` folder.
5. Select the reference, e.g. `SYSPROSRSClientLibrary.dll` and click on **Add**.
6. Ensure the assembly is checked in the list of assemblies and select **OK**.

Affected business objects

The following indicates the business objects that are affected by this feature:

Query objects

SO Document Details Query

The **SO DOCUMENT DETAILS QUERY**[1] business object is used by the Espresso Application (or third party program) to obtain sales order and invoice details. This then returns a list of documents that can be printed.

SO Document Print Query

The **SO DOCUMENT PRINT QUERY**[2] business object is used by the Espresso Application (or third party program) to obtain and return information in the XML format required to generate SRS documents.

[1] Business object: SORQD2

[2] Business object: SORQDP

Referencing

API Methods

The exposed API methods can be used in a number of ways:

- Authentication.
- Controlling login and logout.
- Controlling the formats of the documents.

This lets customers with login credentials access their data to generate documentation from third-party applications. External or third-party applications like ***SYSPRO Espresso*** can request the information from the API and, once received, apply it to generate the required documents.

The `SYSPROSRSClientLibrary.dll` assembly (located in the `\Base` folder) allows an application like ***SYSPRO Espresso*** to call the **SYSPRO 8 Reporting Host Service** in order to create the documents.

The following API methods or functions are exposed:

AuthSysproUser

This controls and authorizes the logging on of users. This method returns a session ID.

Because the API logs on via the service, which is a trusted application, SYSPRO doesn't require a password. Although the method requests passwords, these aren't actually used.

This means that you need to make sure that any operator log on authentication is handled by the third party or custom application.

AuthenticateSysproGUID

This controls and authorizes access of users that are already logged in. This method returns a session ID.

This takes a session ID you have generated by doing a logon using the **SYSPRO 8 e.net Communications Load Balancer** and authenticates that as a valid session ID and returns a GUID.

LogOffUserSession

This logs off sessions of stand-alone applications.

This takes the GUID that is returned by the `AuthSysproUser` function and performs a log off.

This GUID that is returned is not the same as the session ID that is returned from a log on.

DetermineDocument Options

This provides a list of formats for the selected document type as well as sometimes additional information pertaining to the selected document. This method returns XML with details of available documents and formats.

This only returns SRS formats and looks at the control file (i.e. sales order control file), to determine which formats are defined for SRS according to the governing business rules.

This method returns the following that can be used to determine which document types can be printed:

- For the **Sales Order Document Printing** functionality:
  - Sales order number
  - Sales order status
  - Sales order flags
  - Dispatch note number
  - Print & reprint flags
- For other document types:
  - Purchase order number
  - Quote number
  - Factory documentation number

ProduceDocument

This generates the document to print on the host server.

When previewing the document, it is returned in a HEX encoded format of a PDF document

This content must be converted back into ASCII format, before saving it to disk.

The input XML is passed to the business object, which in turn communicates with the **Document Print** program. The **SYSPRO 8 Reporting Host Service** uses the XML returned from the **Document Print** program, to generate the document using the selected format.

Sample XML per document type

Determine document options

API method name: Determine document options

Sales Orders

This function can be used to return information on a specific sales order as well as the formats that have been configured per sales order document type. This allows the application developer to present a user interface to their customer and control which documents can be printed and which print options are available.

For example, if both the `CanPrintAcknowledgement` and `CanRePrintAcknowledgement` values are `FALSE`, then no order acknowledgment can be generated or reprinted for the sales order.

Information for any invoices that have been generated for a sales order will be returned and if there are any lines available to invoice on the sales order then the `CanGenerateInvoice` element will have a value of `TRUE`.

Various document types will return flags to indicate whether a document can be printed or reprinted. It is up to the 3rd party application to read and store these flags and then provide the correct value for `Reprint` element in the produce document input XML ,where applicable.

- The following codes are used for the sales order documents types:
  - `I` - invoice
  - `D` - delivery note
  - `O` - order acknowledgment
  - `N` - dispatch notes

XML In

```
<?xml version="1.0" encoding="windows-1252" ?>
<Query>
  <Option>
    <Function>GETDOCDET</Function>
  </Option>
  <Filter>
    <OrderNumber FilterType='S' FilterValue='791' />
  </Filter>
</Query>
```

XML Out

```
<?xml version="1.0" encoding="windows-1252" ?>
<DocumentControl
Language="05" Language2="EN" CssStyle="" DecFormat="1" DateFormat="01" Role="01" Version="8.0.001" OperatorPrimaryRole="
">
  <DocumentInformation>
    <SalesOrder>000791</SalesOrder>
    <OrderStatus>9</OrderStatus>
    <Translated_OrderStatus>Complete</Translated_OrderStatus>
```

```
    <ActiveFlag>N</ActiveFlag>
    <Translated_ActiveFlag>No</Translated_ActiveFlag>
    <CancelledFlag />
    <Translated_CancelledFlag>No</Translated_CancelledFlag>
    <SODocumentType>O</SODocumentType>
    <Translated_SODocumentType>Order</Translated_SODocumentType>
    <CanPrintAcknowledgement>false</CanPrintAcknowledgement>
    <CanRePrintAcknowledgement>false</CanRePrintAcknowledgement>
    <CanPrintDeliveryNote>false</CanPrintDeliveryNote>
    <CanRePrintDeliveryNote>false</CanRePrintDeliveryNote>
    <CanGenerateInvoice>false</CanGenerateInvoice>
    <Documents>
      <Invoice>
        <InvoiceNumber>100506</InvoiceNumber>
        <InvoiceSource>O</InvoiceSource>
        <Translated_InvoiceSource>Order</Translated_InvoiceSource>
        <DateLastInvPrt>2015-04-08</DateLastInvPrt>
      </Invoice>
    </Documents>
    <Formats>
      <Format>
        <DocumentType>Order Acknowledgement</DocumentType>
        <FormatCode>0</FormatCode>
        <FormatName>Order Acknowledgemen</FormatName>
      </Format>
      <Format>
        <DocumentType>Delivery Note</DocumentType>
        <FormatCode>0</FormatCode>
        <FormatName>Delivery Note</FormatName>
      </Format>
      <Format>
        <DocumentType>Invoice</DocumentType>
        <FormatCode>0</FormatCode>
        <FormatName>Invoice</FormatName>
      </Format>
    </Formats>
  </DocumentInformation>
</DocumentControl>
```

Accounts Receivable statement print

The only options available for this document type is to return a list of the SRS formats that have been configured.

XML In

```
<?xml version="1.0" encoding="windows-1252" ?>
<Query>
  <Option>
    <Function>GETFMTS</Function>
    <Format />
  </Option>
</Query>
```

XML Out

```
<?xml version="1.0" encoding="Windows-1252"?>
<DocumentControl
Language="05" Language2="EN" CssStyle="" DecFormat="1" DateFormat="01" Role="01" Version="8.0.001" OperatorPrimaryRole="
">
  <DocumentInformation>
    <DocumentType>A/R Statement Print</DocumentType>
    <Format>
      <FormatCode>01</FormatCode>
      <FormatName>Customer Statement</FormatName>
    </Format>
  </DocumentInformation>
</DocumentControl>
```

Purchase orders (foreign and local)

This function returns information for a given purchase order as well as all the formats that have been defined so that the user interface can prompt the user to select a format and print. As with the sales order document type, flags are returned to indicate whether the document can be printed and reprinted.

XML In

```
<?xml version="1.0" encoding="Windows-1252"?>
<Query>
  <Option>
    <Function>GETDOCDET</Function>
    <IncludeFormatDetails>Y</IncludeFormatDetails>
  </Option>
  <Filter>
    <PurchaseOrder FilterType="S" FilterValue="420" />
  </Filter>
</Query>
```

XML Out

```
<?xml version="1.0" encoding="Windows-1252"?>
<DocumentControl
Language="05" Language2="EN" CssStyle="" DecFormat="1" DateFormat="01" Role="01" Version="8.0.000" OperatorPrimaryRole="
">
  <DocumentInformation>
    <PurchaseOrder>000420</PurchaseOrder>
    <PurchaseOrderStatus>4</PurchaseOrderStatus>
    <Translated_OrderStatus>Order printed</Translated_OrderStatus>
    <PurchaseOrderType>I</PurchaseOrderType>
    <Translated_PurchaseOrderType>Import order</Translated_PurchaseOrderType>
    <PurchaseOrderActiveFlag />
    <PurchaseOrderCancelledFlag />
    <CanPrintPurchaseOrder>false</CanPrintPurchaseOrder>
    <CanRePrintPurchaseOrder>true</CanRePrintPurchaseOrder>
    <Formats>
      <Format>
        <DocumentType>F</DocumentType>
        <DocumentTypeDescription>Purchase orders - foreign</DocumentTypeDescription>
        <FormatCode>0</FormatCode>
        <FormatName>P/order - Foreign</FormatName>
      </Format>
      <Format>
        <DocumentType>L</DocumentType>
        <DocumentTypeDescription>Purchase orders - Local</DocumentTypeDescription>
        <FormatCode>0</FormatCode>
        <FormatName>P/order - Local</FormatName>
      </Format>
    </Formats>
  </DocumentInformation>
</DocumentControl>
```

Quotations

This function returns information for a given quotation as well as all the formats that have been defined so that the user interface can prompt the user to select a format and print. As with the sales order document type, flags are returned to indicate whether the document can be printed and reprinted.

XML In

```
<?xml version="1.0" encoding="Windows-1252"?>
<Query>
  <Option>
    <Function>GETDOCDET</Function>
    <IncludeFormatDetails>Y</IncludeFormatDetails>
  </Option>
  <Filter>
    <Quotation FilterType="S" FilterValue="344" />
  </Filter>
</Query>
```

XML Out

```
<?xml version="1.0" encoding="Windows-1252"?>
<DocumentControl
Language="05" Language2="EN" CssStyle="" DecFormat="1" DateFormat="01" Role="01" Version="8.0.001" OperatorPrimaryRole="
">
  <DocumentInformation>
    <Quotation>00000344</Quotation>
    <QuotationVersion>000</QuotationVersion>
    <QuoteStatus>0</QuoteStatus>
    <Translated_OrderStatus>In progress</Translated_OrderStatus>
    <CanPrintQuotation>false</CanPrintQuotation>
    <CanRePrintQuotation>false</CanRePrintQuotation>
    <Formats>
      <Format>
        <FormatCode>0</FormatCode>
        <FormatName>Quotation</FormatName>
      </Format>
      <Format>
        <FormatCode>1</FormatCode>
        <FormatName>Quotation - Multiple</FormatName>
      </Format>
    </Formats>
  </DocumentInformation>
</DocumentControl>
```

Factory documentation

The only options available for this document type is to return a list of the SRS formats that have been configured.

XML In

```
<?xml version="1.0" encoding="Windows-1252"?>
<Query>
  <Option>
    <Function>GETFMTS</Function>
  </Option>
</Query>
```

XML Out

```
<?xml version="1.0" encoding="Windows-1252"?>
<DocumentControl
Language="05" Language2="EN" CssStyle="" DecFormat="1" DateFormat="01" Role="01" Version="8.0.001" OperatorPrimaryRole="
">
  <DocumentInformation>
    <Format>
      <DocumentNumber>1</DocumentNumber>
      <FormatCode>1</FormatCode>
      <FormatName>Job Card</FormatName>
    </Format>
    <Format>
      <DocumentNumber>2</DocumentNumber>
      <FormatCode>3</FormatCode>
      <FormatName>Picking Slip</FormatName>
    </Format>
  </DocumentInformation>
</DocumentControl>
```

Produce document

API method name: ProduceDocument

The XML parameters for each document type are the same, the only difference would be the `DocumentType` element which indicates for which document type you are generating the document. The `DocumentType` element is not case sensitive.

The purpose of the parameter XML is to tell the function what you would like to do with the document that is generated.

Sales Orders

XML Parameters

```
<?xml version="1.0" encoding="windows-1252" ?>
<DocumentControl>
  <DocumentType>SalesOrder</DocumentType>
  <Print>False</Print>
  <Email>False</Email>
  <Preview>True</Preview>
  <XmlOnly>False</XmlOnly>
  <PrinterDetails>
    <PrinterName />
    <PrintCopies>1</PrintCopies>
    <PrintCollate>True</PrintCollate>
  </PrinterDetails>
  <EmailDetails>
    <EmailFromAddress />
    <EmailToAddress />
    <EmailCCAddress />
    <EmailToAddress />
    <EmailBodyText />
  </EmailDetails>
</DocumentControl>
```

XML In

When printing a sales order, you need to supply the `DocumentType` element to tell the API which sales order document you want to produce, in this case it is an invoice.

FOR EXAMPLE:

This instructs the API to reprint invoice 100506 for sales order 791 using format 0.

```
<?xml version="1.0" encoding="windows-1252" ?>
<Query>
  <Option>
    <Function>ONLINE</Function>
    <DocumentType>I</DocumentType>
    <Format>0</Format>
    <Reprint>Y</Reprint>
    <PrintDispatchInvoice>N</PrintDispatchInvoice>
  </Option>
  <Filter>
    <OrderNumber FilterType='S' FilterValue='000791' />
    <InvoiceNumber FilterType='S' FilterValue='100506' />
  </Filter>
</Query>
```

FOR EXAMPLE:

This instructs the API to reprint dispatch note invoice 100540 for sales order 945 and dispatch note 0000029 using format 0.

```
<?xml version="1.0" encoding="windows-1252" ?>
<Query>
  <Option>
    <Function>ONLINE</Function>
    <DocumentType>I</DocumentType>
    <Format>0</Format>
    <Reprint>Y</Reprint>
    <PrintDispatchInvoice>Y</PrintDispatchInvoice>
  </Option>
  <Filter>
    <InvoiceNumber FilterType="S" FilterValue="100540" />
    <OrderNumber FilterType="S" FilterValue="000000000000945" />
    <DispatchNote FilterType="S" FilterValue="000000000000027" />
  </Filter>
</Query>
```

XML Out

```
<?xml version="1.0" encoding="windows-1252" ?>
<Query>
  <Document>
    <Status />
    <DocumentGuid/>
    <ErrorMessage/>
    <StatusPreview/>
    <DocumentHex>HEX Encoded PDF file</DocumentHex>
  </Document>
</Query>
```

Accounts Receivable statement print

XML Parameters

```
<?xml version="1.0" encoding="windows-1252" ?>
<DocumentControl>
  <DocumentType>AR Statement Print</DocumentType>
  <Print>False</Print>
  <Email>False</Email>
  <Preview>True</Preview>
  <XmlOnly>False</XmlOnly>
  <PrinterDetails>
    <PrinterName />
    <PrintCopies>1</PrintCopies>
    <PrintCollate>True</PrintCollate>
  </PrinterDetails>
  <EmailDetails>
    <EmailFromAddress />
    <EmailToAddress />
    <EmailCCAddress />
    <EmailToAddress />
    <EmailBodyText />
  </EmailDetails>
</DocumentControl>
```

XML In

The AR Statement format allows for multiple options to be provided via a 3rd party API.

For more information on these elements, please refer to the **AR Statement Print** program in SYSPRO

```
<?xml version="1.0" encoding="Windows-1252"?>
<Query>
  <Option>
    <Function>ONLINE</Function>
    <Format>0</Format>
    <StatementAsOf>C</StatementAsOf>
    <StatementDate />
    <StatementAgeing>S</StatementAgeing>
    <ConsolidateSub>N</ConsolidateSub>
    <IncludeAttached>N</IncludeAttached>
    <AsOpenItem>N</AsOpenItem>
    <BalanceType>A</BalanceType>
    <MinimumBalance>0.00</MinimumBalance>
    <SalesMessage />
  </Option>
  <Filter>
    <Customer FilterType="S" FilterValue="0000001" />
  </Filter>
</Query>
```

XML Out

```
<?xml version="1.0" encoding="windows-1252" ?>
<Query>
  <Document>
    <Status />
    <DocumentGuid/>
    <ErrorMessage/>
    <StatusPreview/>
    <DocumentHex>HEX Encoded PDF file</DocumentHex>
  </Document>
</Query>
```

Purchase orders (foreign and lcoal)

XML Parameters

Supported values for the `DocumentType` element are `PURCHASE ORDERS - LOCAL` and `PURCHASE ORDERS - FOREIGN`.

```
<?xml version="1.0" encoding="windows-1252" ?>
<DocumentControl>
  <DocumentType>Purchase orders - Local</DocumentType>
  <Print>False</Print>
  <Email>False</Email>
  <Preview>True</Preview>
  <XmlOnly>False</XmlOnly>
  <PrinterDetails>
    <PrinterName />
    <PrintCopies>1</PrintCopies>
    <PrintCollate>True</PrintCollate>
  </PrinterDetails>
  <EmailDetails>
    <EmailFromAddress />
    <EmailToAddress />
    <EmailCCAddress />
    <EmailToAddress />
    <EmailBodyText />
  </EmailDetails>
</DocumentControl>
```

XML In

Valid values for the `DocumentType` element are either `L` (local) or `F` (foreign) and the format code selected must have been created for the specified purchase order document type.

```
<?xml version="1.0" encoding="Windows-1252"?>
<Query>
  <Option>
    <Function>ONLINE</Function>
    <DocumentType>L</DocumentType>
    <Format>0</Format>
    <Reprint>Y</Reprint>
  </Option>
  <Filter>
    <OrderNumber FilterType="S" FilterValue="458" />
  </Filter>
</Query>
```

XML Out

```
<?xml version="1.0" encoding="windows-1252" ?>
<Query>
  <Document>
    <Status />
    <DocumentGuid/>
    <ErrorMessage/>
    <StatusPreview/>
    <DocumentHex>HEX Encoded PDF file</DocumentHex>
  </Document>
</Query>
```

Quotations

Supported values for the `DocumentType` element are `QUOTATION - SINGLE`, `QUOTATION - MULTIPLE LINE` and `QUOTATION - MULTIPLE COLUMN`.

XML Parameters

```
<?xml version="1.0" encoding="windows-1252" ?>
<DocumentControl>
  <DocumentType>Quotation - Single</DocumentType>
  <Print>False</Print>
  <Email>False</Email>
  <Preview>True</Preview>
  <XmlOnly>False</XmlOnly>
  <PrinterDetails>
    <PrinterName />
    <PrintCopies>1</PrintCopies>
    <PrintCollate>True</PrintCollate>
  </PrinterDetails>
  <EmailDetails>
    <EmailFromAddress />
    <EmailToAddress />
    <EmailCCAddress />
    <EmailToAddress />
    <EmailBodyText />
  </EmailDetails>
</DocumentControl>
```

XML In

```
<?xml version="1.0" encoding="Windows-1252"?>
<Query>
  <Option>
    <Function>ONLINE</Function>
    <Format>0</Format>
    <Reprint>Y</Reprint>
    <DocumentDate />
    <PrintDefaultOffer />
    <PrintOffer1>Y</PrintOffer1>
    <PrintOffer2 />
    <PrintOffer3 />
    <PrintOffer4 />
    <PrintOffer5 />
  </Option>
  <Filter>
    <Quotation FilterType="S" FilterValue="951" />
  </Filter>
</Query>
```

XML Out

```
<?xml version="1.0" encoding="windows-1252" ?>
<Query>
  <Document>
    <Status />
    <DocumentGuid/>
    <ErrorMessage/>
    <StatusPreview/>
    <DocumentHex>HEX Encoded PDF file</DocumentHex>
  </Document>
</Query>
```

Factory documentation

XML Parameters

```
<?xml version="1.0" encoding="windows-1252" ?>
<DocumentControl>
  <DocumentType>Factory Documentation</DocumentType>
  <Print>False</Print>
  <Email>False</Email>
  <Preview>True</Preview>
  <XmlOnly>False</XmlOnly>
  <PrinterDetails>
    <PrinterName />
    <PrintCopies>1</PrintCopies>
    <PrintCollate>True</PrintCollate>
  </PrinterDetails>
  <EmailDetails>
    <EmailFromAddress />
    <EmailToAddress />
    <EmailCCAddress />
    <EmailToAddress />
    <EmailBodyText />
  </EmailDetails>
</DocumentControl>
```

XML In

```
<?xml version="1.0" encoding="Windows-1252"?>
<Query>
  <Option>
    <DocumentNumber>1</DocumentNumber>
    <Format>0</Format>
  </Option>
  <Filter>
    <Job FilterType="S" FilterValue="951" />
  </Filter>
</Query>
```

XML Out

```
<?xml version="1.0" encoding="windows-1252" ?>
<Query>
  <Document>
    <Status />
    <DocumentGuid/>
    <ErrorMessage/>
    <StatusPreview/>
    <DocumentHex>HEX Encoded PDF file</DocumentHex>
  </Document>
</Query>
```

Sample code using the sales order document types

The following sample code is provided to assist you in using the API.

The sample code is provided in C# and is specifically for sales order document types. It needs to be adjusted if used for other document types.

Creating the SYSPROSRSClient object

```
// Replace "localhost:20140" with the endpoint of your
// SRS reporting host service
// See Reporting service field on the Reporting tab in System Setup
SYSPROSRSClientLibrary.SYSPROSRSClient _sysproSRSClient =
new SYSPROSRSClientLibrary.SYSPROSRSClient("localhost:20140")
```

Authenticate to the service using operator and company code

The passwords are not validated using this function.

The custom or third party application must handle authentication, should this be required.

```
// The following assumes that you have set up the OperatorCode and
// CompanyCode method arguments with appropriate values.
// Passwords are not required as the host service is a trusted application
string sessionId = "";sessionId = _sysproSRSClient.AuthenticateSYSPROUser(OperatorCode, "",
                                                                          CompanyCode, "");
```

Authenticate to the service using Logon GUID

You would only use this function if you already have a session ID that you want to use.

This method requires operator and company password.

```
// // Create the Load Balancer instance using the e.Net Load Balancer end point
// See Server name and REST port fields on the e.net service details tab
//   in System Setup
// Requires "using SYSPROWCFServicesClientLibrary40;"
SYSPROWCFServicesPrimitiveClient WCFNETTCP;
WCFNETTCP = new SYSPROWCFServicesPrimitiveClient(WCFAddress,
                                                 SYSPROWCFBinding.NetTcp);

// Generate the session id with the Logon function to the relevant
//   SYSPRO instance
string GUID = WCFNETTCP.Logon(OperatorCode, OperatorPassword,
                              CompanyCode, CompanyPassword,
                              "", "", SYSPROInstance, "");

// Validate the session id with the API
string sessionId = "";
sessionId = _sysproSRSClient.AuthenticateSYSPROGuid(GUID);
```

Determine the available document options

```
// Validate the session id with the API
sessionId = _sysproSRSClient.AuthenticateSYSPROGuid(GUID);
StringBuilder xmlIn = new StringBuilder();
xmlIn.Append("<Query>");
xmlIn.Append("<Option>");
xmlIn.Append("<Function>GETDOCDET</Function>");
xmlIn.Append("<IncludeFormatDetails>Y</IncludeFormatDetails>");
xmlIn.Append("</Option>");
xmlIn.Append("<Filter>");
xmlIn.AppendFormat("<OrderNumber FilterType='S' FilterValue='{0}' />", 791);
// Where 791 is the Sales Order number
xmlIn.Append("</Filter>");
xmlIn.Append("</Query>");
string documentOptions = _sysproSRSClient.DetermineDocumentOptions(
sessionId, "SalesOrder", xmlIn.ToString());
// See documentation for sample XML out
```

Produce a document for sales order invoice

```
Produce a document for sales order invoiceStringBuilder xmlParam = new StringBuilder();
xmlParam.Append("<DocumentControl>");
xmlParam.Append("<DocumentType>SalesOrder</DocumentType>");
xmlParam.Append("<Print>False</Print>");
xmlParam.Append("<Preview>True</Preview>");
xmlParam.Append("<XmlOnly>False</XmlOnly>");
xmlParam.Append("<PrinterDetails>");
xmlParam.Append("<PrinterName />");
xmlParam.Append("<PrintCopies>1</PrintCopies>");
xmlParam.Append("<PrintCollate>True</PrintCollate>");
xmlParam.Append("</PrinterDetails>");
xmlParam.Append("<EmailDetails>");
xmlParam.Append("<EmailFromAddress />");
xmlParam.Append("<EmailToAddress />");
xmlParam.Append("<EmailCcAddress />");
xmlParam.Append("<EmailSubject />");
xmlParam.Append("<EmailBodyText />");
xmlParam.Append("</EmailDetails>");
xmlParam.Append("</DocumentControl>");
StringBuilder xmlIn = new StringBuilder();
xmlIn.Append("<Query>");
xmlIn.Append("<Option>");
// This is always 'ONLINE'
xmlIn.Append("<Function>ONLINE</Function>");
// See Document Type Codes
xmlIn.Append("<DocumentType>I</DocumentType>");
// Format code selected to print with - see output from DetermineDocumentOptions
xmlIn.Append("<Format>0</Format>");
xmlIn.Append("<Reprint>Y</Reprint>");
Print/Reprint flag - see output from DetermineDocumentOptions
xmlIn.Append("</Option>");
xmlIn.Append("<Filter>");
xmlIn.AppendFormat("<OrderNumber FilterType='S' FilterValue='{0}' />",791);
//Where 791 is the Sales Order number
xmlIn.AppendFormat("<InvoiceNumber FilterType='S' FilterValue='{0}' />",100506);
// Where 100506 is the Invoice number
xmlIn.Append("</Filter>");
xmlIn.Append("</Query>");
string xmlOut = _sysproSRSClient.ProduceDocument(
sessionId, xmlParam.ToString(), xmlIn.ToString());
// See documentation for sample XML out
```

Retrieve the document content from the HEX encoded string from the output XML

```
byte[] docByte = GetByteStringFromOutput(xmlOut);
/// <summary>
/// Function that converts a Hex encoded string to an array
/// of unsigned integers that represents the ASCII bytes
/// </summary>
private static byte[] GetByteStringFromOutput(string XMLOut)
{
 byte[] bytes = new byte[] { };
try
{
XDocument XOut = XDocument.Parse(XMLOut);
// Use Linq to get the HEX encoded document string
XElement docHex = XOut.Descendants().Where(n =>
                  n.Name == "DocumentHex").First();
if (docHex == null)
{
return bytes;
}
string HexString = docHex.Value;
// Get the number of characters in the string
int NumberChars = HexString.Length;
// Each byte is derived from 2 characters in the input
// that together represent the HEX value of the byte
bytes = new byte[NumberChars / 2];
for (int i = 0; i < NumberChars; i += 2)
{
// The 16 in the following conversion indicates that this
// is converting from HEX, or Base 16.
bytes[i / 2] = Convert.ToByte(HexString.Substring(i, 2), 16);
}
return bytes;
}
catch { throw; }
}
```

Logoff from the SYSPRO session

```
// If you authenticated using a OperatorCode and
// CompanyCode then you might want to logoff.
_sysproSRSClient.LogoffUserSession(sessionId);
```

SYSPRO™

www.syspro.com