SunSystems Connect (SSC)

For SunSystems 5.2.1 SP1

SunSystems 5 – SSC Training Notes.

SunSystems Document version 1.0 – June 2005.

Written by Michael Tendler.

Based on SunSystems software version 5.2.1 SP1.

Systems Union Holdings Ltd
Systems Union House
1 Lakeside Road
Aerospace Centre
Farnborough
Hampshire
GU14 6XP

United Kingdom

Course Contents

Chapter 1 - Overview

What is SSC?

SunSystems Connect (SSC) is an extendable Java-based open integration tool that enables you to link with SunSystems programmatically via. an exposed interface.

It allows 3rd party applications to gain access to SunSystems data in a controlled manner, taking advantage of the full power of the underlying business logic. The applications can query, insert, and update SunSystems 5 data and submit commercial transactions in real time.

Interfaces

SSC provides access to SunSystems functionality using standard web services interfaces (SOAP and WSDL). These interfaces support XML input.

Historically SSC also had a native Java RMI interface, although this is no longer supported.

SSC exposes data and functionality through SunSystems components. This means any application that can read or write XML and communicate with one of the published interfaces can link to SunSystems 5 application components.

Components & Methods

An SSC component provides a set of methods to operate with SunSystems data. Generally, a component represents a given entity within SunSystems itself such as Customer, Chart Of Accounts, or Ledger, etc. From a programming perspective can be thought of to some extent as objects, which contain their relevant functions, or methods. For example, the Address component exposes methods to create, amend, query, and delete entries in the address tables.

A full set of SSC components is installed with SunSystems. We will discuss how to view available components and their methods later.

Below is another example of a component and it's associated methods:

Component	**Method**	**Description**
PurchaseOrder	ConfirmLineOrder	*Confirms a line or an entire order*
	CreateOrAmend	*Create a new Purchase Order or amend an existing Purchase Order in SunSystems, and report it in an xml payload*
	DeleteLineOrder	*Deletes either a line or an entire order*
	DeleteLines	*Delete specified lines of an order*
	DeleteWholeOrder	*Deletes the whole order*
	HoldLineOrder	*Holds a line or an entire order*
	Query	*Extract the Purchase Orders from SunSystems*
	ReleaseLineOrder	*Release a Purchase Order Line*

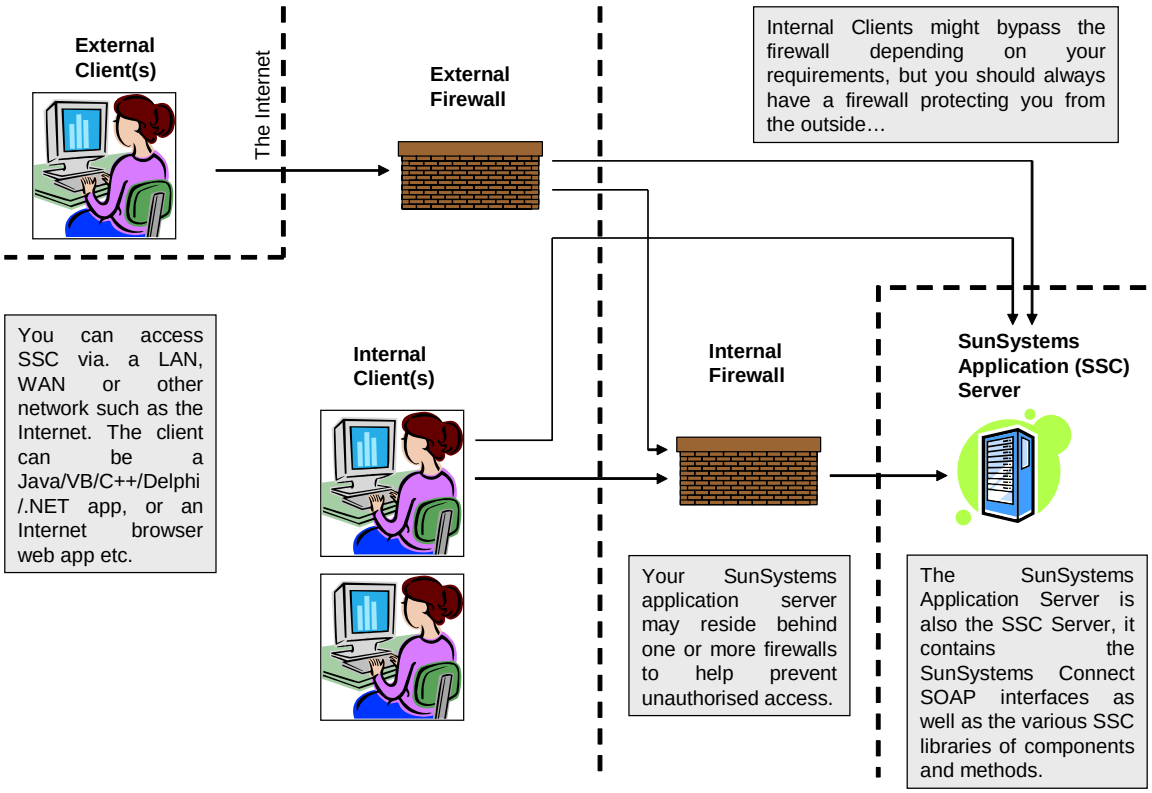
Chapter 2 - Architecture

Client / Server

The SSC technology is a real-time interface and resides on the same physical machine as the SunSystems Application Service Layer. As it is exposed as a SOAP interface, any 3rd party applications wishing to communicate with it must look to the SunSystems Application server as the SSC Server.

It is the responsibility of the 3rd party developer to ensure their application can handle loss of connectivity due to network problems or incorrect configuration. This may involve a queuing mechanism or other storage of data to be sent to SunSystems while the link is down.

The architecture of connectivity to SSC can be illustrated as follows:



Tomcat

The Tomcat server is a Java based Web Application container that was created to run Servlets and JavaServer Pages (JSP) in Web applications. SSC is hosted by Tomcat.

Drivers

There are several ways in which a method of a component can do its work. Each of these ways is referred to as a “driver”.

There are 5 different types of driver used by SSC. Selection of the appropriate driver is controlled internally.

SASI Driver

SASI stands for SunSystems Application Services Interface. The SASI driver operates by using SunSystems 5 business logic (such as user defined rules, validation and database access) during payload execution. The full capabilities of SunSystems business logic are available through the methods implemented by the SASI driver. The SASI driver is configured with a script written in a language similar to Java, describing the interaction between SSC and SunSystems. This driver is only available for selected component methods.

DSI Driver

The DSI operates by using SunSystems 5 business logic (such as user defined business rules, and database access), during payload execution. However, it bypasses the forms interface used by the SASI driver. This makes the driver faster and allows better scalability for enterprise environments. The driver is configured to use the same scripting interface as SASI.

DJI Driver

The Direct Java Interface provides high performance and is primarily intended for bulk payloads containing large numbers of transactions where performance is critical. This is achieved by bypassing the SunSystems business logic processed by the SASI driver and instead replicating those rules internally. As the method no longer has the overhead of maintaining the connection between SSC and the SunSystems function, this enables more transactions to be processed in a given time. This driver is only available for selected component methods.

Important Note: SunSystems Business Rules are not invoked by the DJI Driver, if you want Business Rules to be applied to your data, you must execute your payload with the component method that uses the SASI or DSI driver. Additionally there may be cases where the validation and business logic applied to the data being imported differs slightly from its SASI counterpart. Again, if you find this occurring you may wish to choose to utilise the component method that uses the SASI driver.

Export Driver

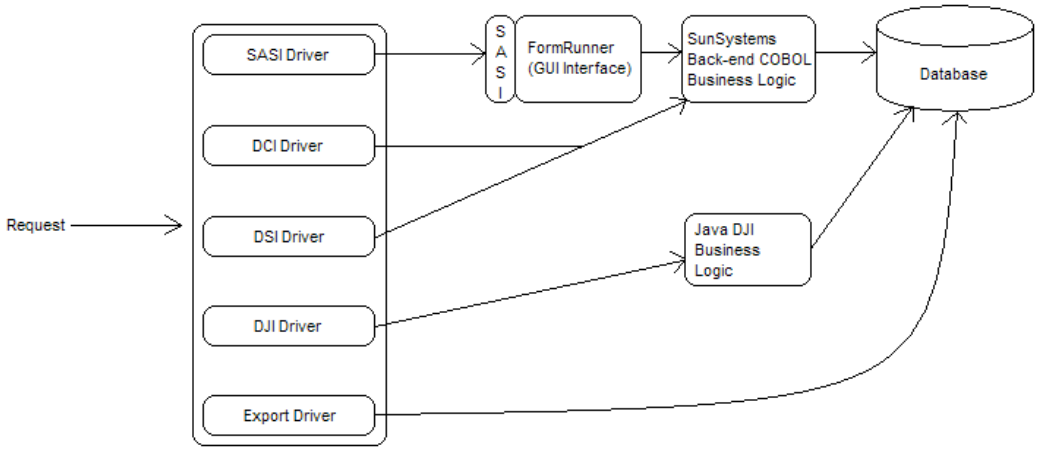
The Export Driver is exclusively used by the Query method and supports direct access to SQL data stored in SunSystems databases.

DCI Driver

The Direct COBOL Interface manages access to COBOL libraries. This functionality has been superseded by DJI and DSI, and the DCI driver will at some point be removed in a future release of SSC.

Note: The DCI driver is mentioned above for completeness and is of historical interest to developers only.

Driver Architecture Diagram



Decoupled Driver Processes

From SunSystems 5.2.1 onwards, SSC provides the facility for decoupling the drivers (for example SASI or DJI), used to execute transactions from the SunSystems Connect service. The service can then be used for its primary job of transaction management, while the logic for each driver executes in a separate process. This allows SSC to process more transactions, reduce response times and to handle more user connections.

In addition, driver process decoupling allows SSC to recycle driver processes more efficiently as memory usage has been improved. This is important if your installation requires continuous availability.

Configuration

You can configure SSC to use decoupled driver processes by using the SunSystems Property Editor (PED) function.

> **Note:** Always make a backup of the <SunSystems>\ssc\Lib\properties \user_props.xml file before you modify any settings. Editing property values can have serious implications for your system and could cause problems that may require you to reinstall SunSystems. We cannot guarantee that problems resulting from the incorrect use of Property Editor can be solved. Use Property Editor at your own risk.

Open Property Editor and find the driver/remote group.

This group contains three sub-groups: dji, export and sasi. These represent the driver types used by SSC when executing transactions.

Each driver type sub-group contains the same three properties: remote, time and timeout.

Properties

Key	`driver/remote/drivertype` where `drivertype` is one of: `sasi`, `export` or `dji`
Property Name	`remote`
Description	This property controls whether the SASI driver should run as a separate process.
Data Type	Boolean
Value Data	Set to true if the driver is to run as a separate process. Set to false if the driver is NOT to run as a separate process. Installation default = false. In previous releases drivers did not run as separate processes.

Key	`driver/remote/drivertype` where `drivertype` is one of: `sasi`, `export` or `dji`
Property Name	`time`
Description	This property sets the absolute time (in 24hour clock format) at which to recycle the driver process.
Data Type	String
Value Data	Minimum value: 00:00 Maximum value: 23:59 There is no installation default
Notes	This property takes precedence over the timeout property if a value for the timeout property exists.

Key	`driver/remote/drivertype` where `drivertype` is one of: `sasi`, `export` or `dji`
Property Name	`timeout`
Description	This setting controls the time interval in minutes for recycling the driver processes.
Data Type	String
Value Data	Minimum value: 10 Maximum value: unlimited Installation default = 360 (6 hours)
Notes	We recommend that you leave this value as the default setting. You may have to experiment to find the best value for your installation as it depends heavily on user load.

Considerations on the use of time and timeout properties

If you wish to set driver recycling for a specific time, set the `time` property value.

If you wish to set driver recycling to happen after a particular time interval has passed, set the `timeout` property value.

Whilst you can have both properties active at the same time, the `time` property takes precedence. For example, you can schedule driver recycling at midnight, or when system activity is low and configure SSC to recycle driver processes every 3 hours, to deliver optimum performance.

Note: Property Editor is discussed in more detail in Chapter 7.

Completing the configuration

Once you have completed setting the values in Property Editor, the new settings are stored in the user_props.xml file and a backup file xxxx.bkp is created automatically.

You must stop and then restart the SunSystems Connect Server service for the settings to take effect.

SSC "Push"

Ordinarily, SSC is a request based entity, listening for instructions which have been initiated by a 3rd party application. Push Processing allows an action happening within specific SunSystems functions, such as the creation or amendment of a supplier address, to automatically trigger a background SSC process. You can use Push Processing to invoke SSC to automatically generate an SSC export XML file containing the new address details. You can then use this XML file as an import to an external XML-aware application.

> **Note:** An action that triggers Push Processing can also be the running of an SSC component, for example, the creation of an address from Transfer Desk or from the SSC web page.

Push Processing allows SunSystems data to be 'pushed' out, so that it can be picked up by an external application whenever the data is created amended or deleted.

> **Note:** It is not possible to push data from SunSystems directly to an external application yet. This means that your application must import the data by starting a process manually, or by an automated process that it provides already. No means is provided in this release to automatically notify an external application of the result of the push process. If you require notification from your external application about whether the push process result succeeded or failed, you must do this in a separate step.

You can set up Push Processing to be invoked whenever data is created, amended or deleted for the following SunSystems functions:

SunSystems Function	SunSystems Shortcut Code
Chart of Accounts Setup	CA
Addresses Setup	AS
Analysis Codes Setup	NC
Analysis Dimensions Setup	ND
Fixed Asset Setup	FAS
Currency Daily Rates Setup	CND
Currency Period Rates Setup	CNP
Customer Setup	CUS
Data Access Groups Setup	DAG
Item Master	IS
Rules Messages Setup	RMS
Rule Set Details	RS
Supplier Setup	SUS

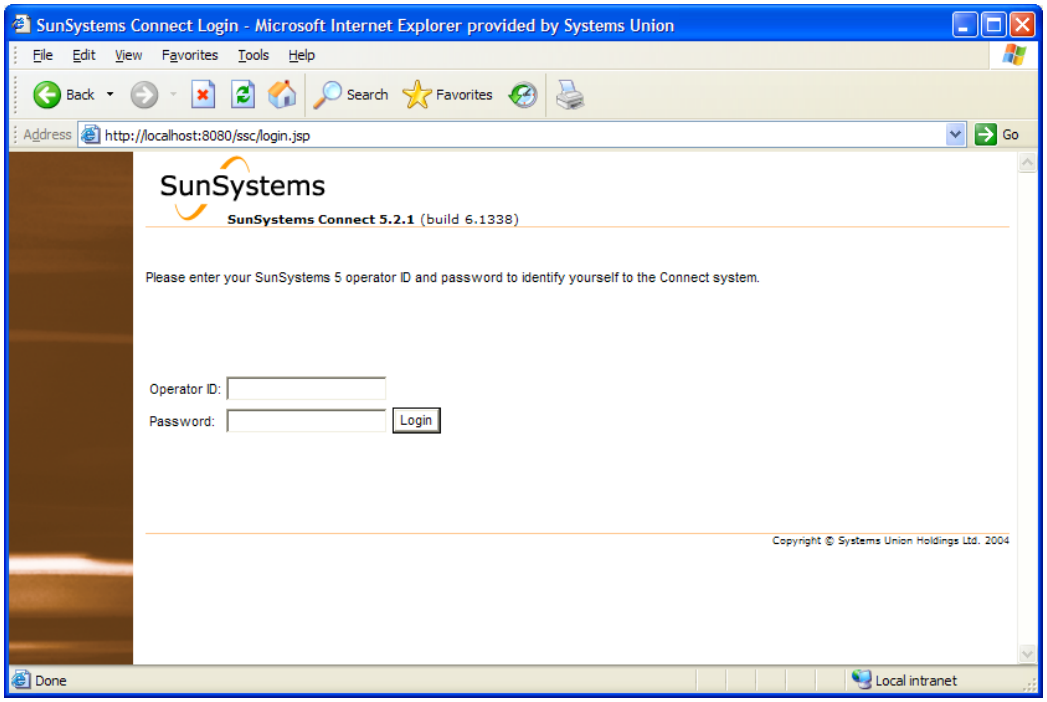
Chapter 3 – SSC Demo Web Page

Overview

To access the page you need to enter the following URL:-

```
http://<sscservername>:8080/ssc/login.jsp.
```

The following web page is displayed:



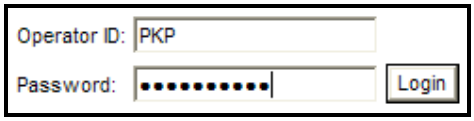
<sscservername> is the name of the SSC server which you wish to 'talk' to. In the example above, SSC and SunSystems were installed on the same machine, hence it is localhost.

"8080" is the port that is being used. This is the Tomcat default, although you can change it if needed. If secure sockets need to be used instead the port setting will be 8443, and you need to specify https:// in the above URL.

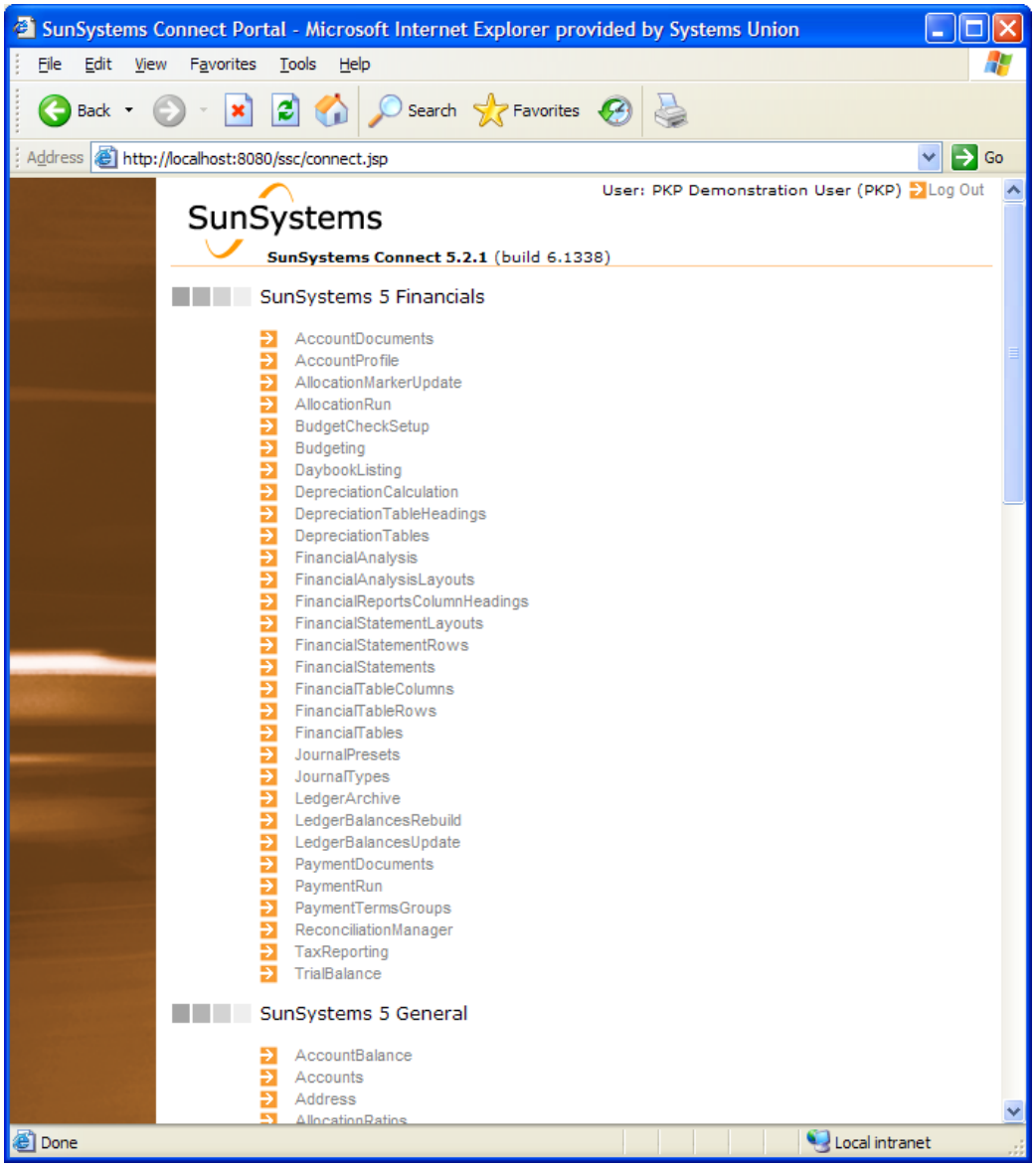
"/ssc/login.jsp" is the virtual path to the .JSP page which contains the Connect Login Web Page. The actual folder name on the server is <SunSystems5>\ssc\tomcat\webapps\ssc.

In order to use the SSC Web Page, you need to first 'log in'. This is for security purposes.

Log in using a valid user name and password:

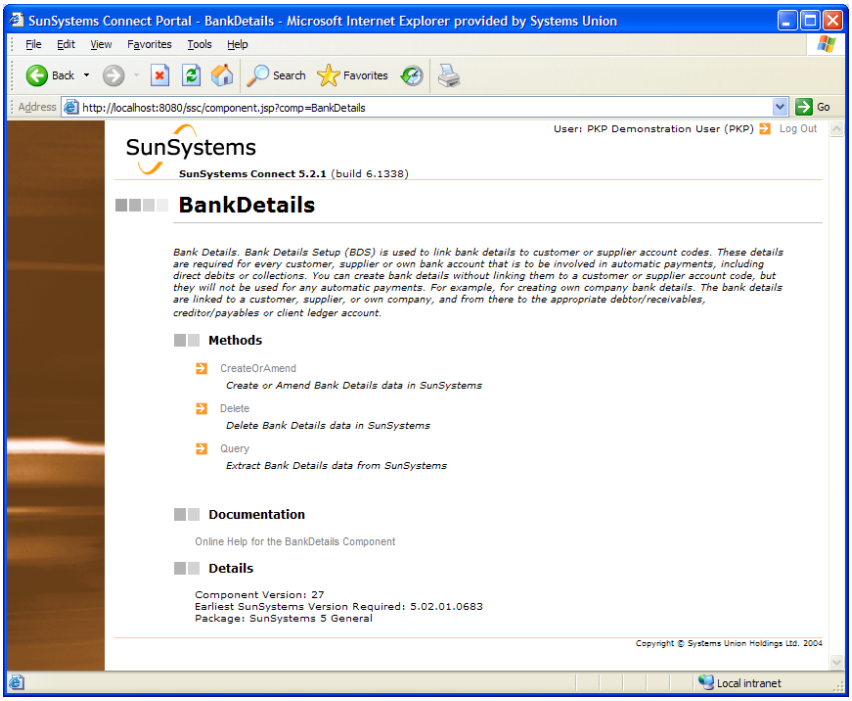


Once in, all the components are listed, sorted by type, and then alphabetically:

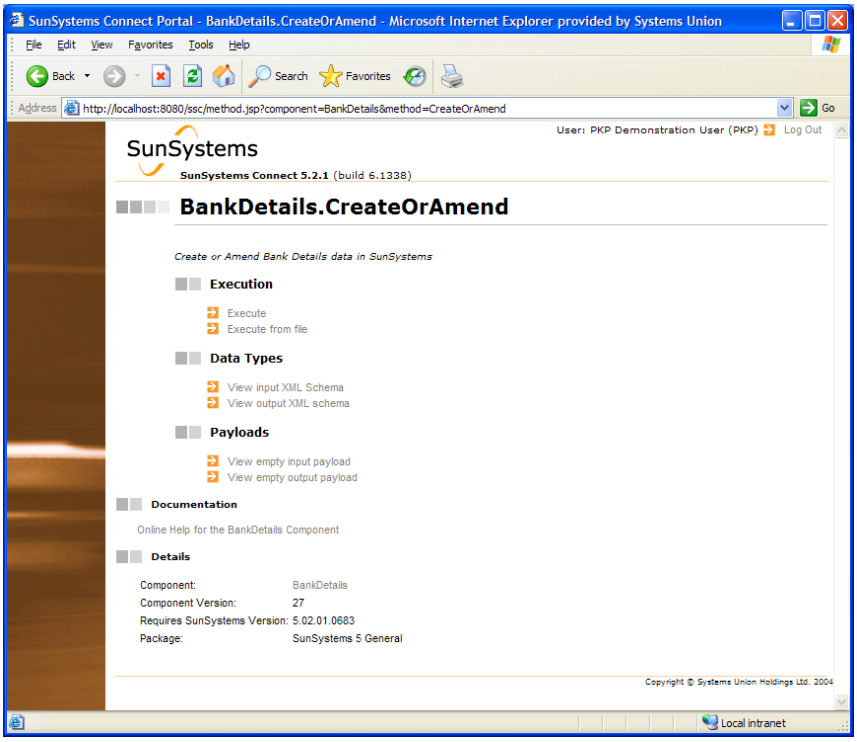


Component specific options

Selecting the desired component reveals a description of the Component along with the methods that can be performed by that component.

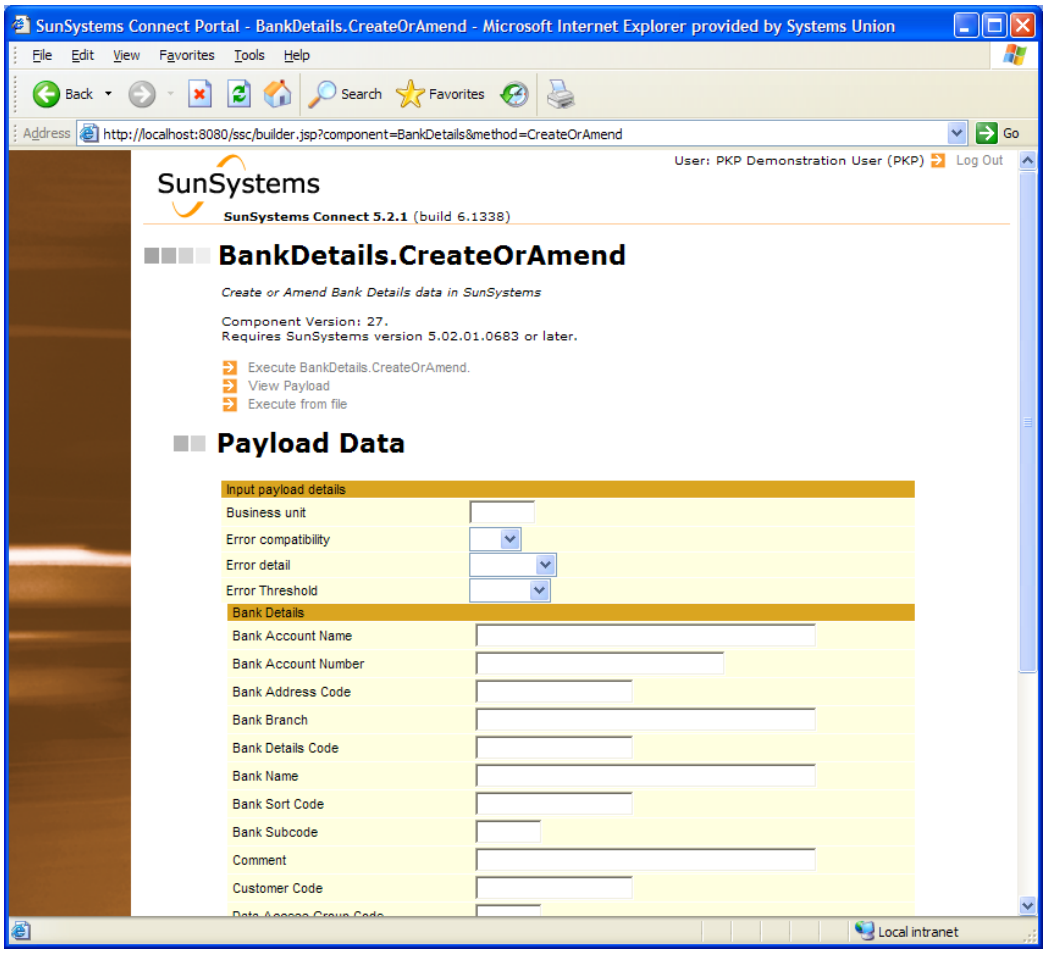


Clicking any of the method names displays a description of the method and the options available:



Execute

This option allows you to enter values on-screen for each field you wish to enter in the input SSC Payload.

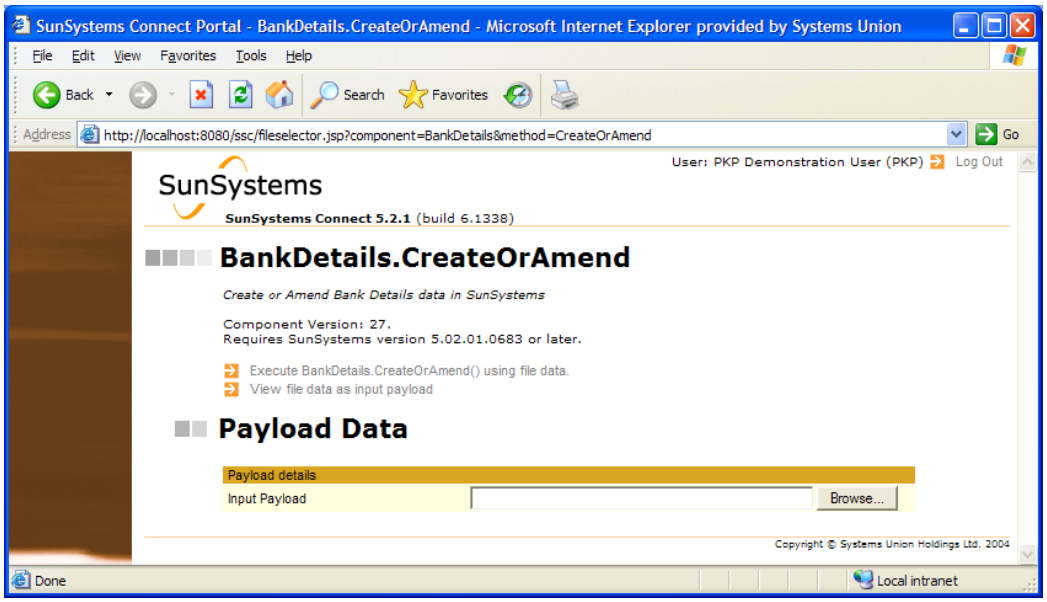


Click on “Execute <*component.method*>” to run the component.

“View Payload” shows you the SSML payload that will be created and sent to the SSC Component.

Execute from file

The “Execute from File” option allows you to run SSC static data updates/queries straight from a saved XML file. This is useful for testing so you can do the query or static data update once and use it again.

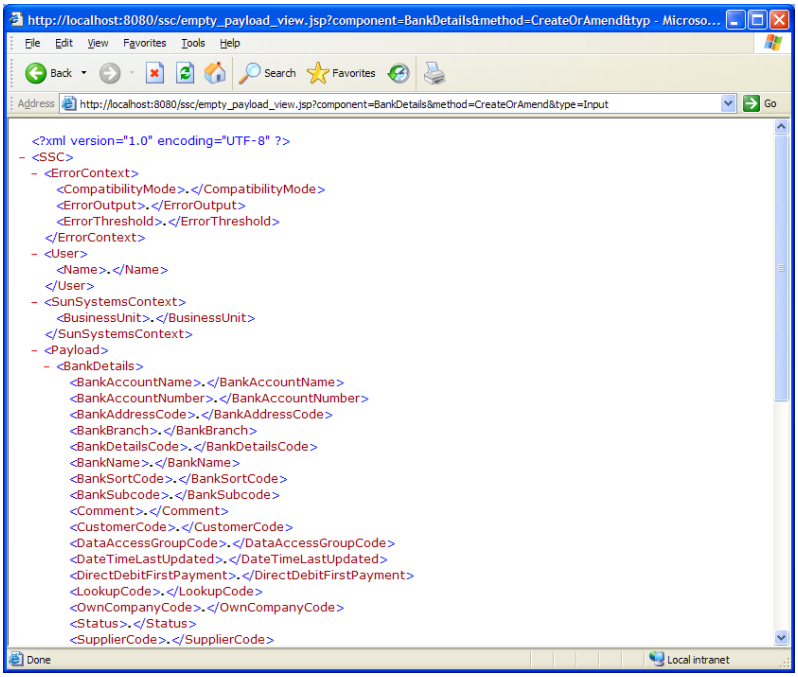


When you select this option, you then need to browse for the required XML file.

Click the “Execute <*component.method*> using file data.” to run the component and method, or click “View file data as input payload” to view the data before executing it.

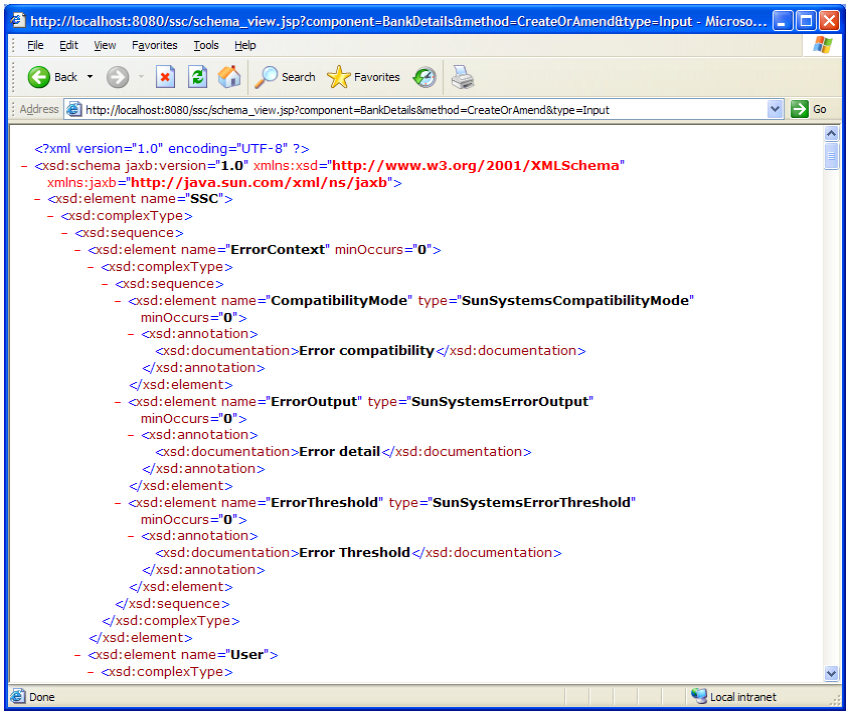
View Empty Input and Output Payload

When one of these functions is selected all the XML tags for that given payload will be displayed (as shown in the screenshot below).

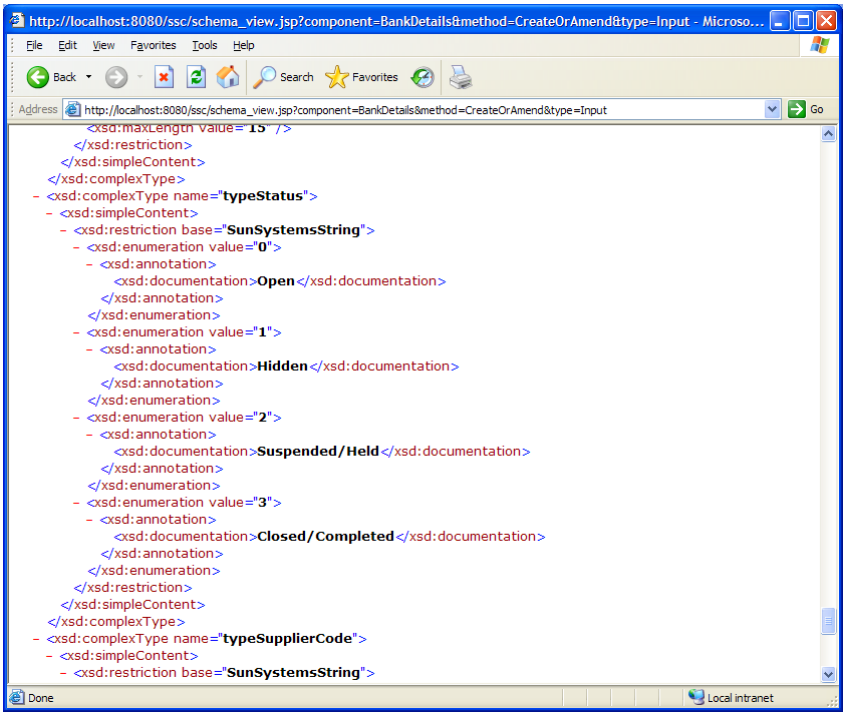


View Input and Output XML Schema

When one of the viewing schema functions is selected, the required schema will be displayed, showing all the XML tags, field sizes, valid values etc.



The most useful part of the Schema from a developers point of view, is the Valid Values for certain fields, which you can find towards the bottom of this document.



You can see from this screenshot the valid values for the Status Code field.

Querying data

When querying data you can add filters to restrict the amount of data returned from SunSystems. Here is a table of the operators and the 'codes' to use:

Meaning	Operator
Equals	=
Not Equal	!=
Greater Than	>
Greater Than Or Equals	>=
Less Than	<
Less Than Or Equals	<=
List	><
Like	~=
Range	<>

This is demonstrated in exercise 3, below.

EXERCISE 1 – Query all Address codes

Load Internet Explorer by clicking the "e" icon on your Desktop: 

Load the SunSystems Connect demo/web page by entering the URL:

http://localhost:8080/ssc/login.jsp.

Login as 'PKP' – there is no password.

Select the "Address" component, in the group "SunSystems 5 General", and then click on "Query". Then from the list which appears, select "Execute".

In the Header section of the resulting screen, enter "PKP" in the Business Unit field.

We will leave the Filter field blank, because we want to retrieve all records (we will do an exercise on filters soon).

Place a Tick in the "Address Code" field, as shown:

Click on the "View Payload" option at the top to see the SSML which will automatically be generated, and sent to SSC.

Click the browsers "back" button to return to the entry screen. Re-enter the data, and click on the "Execute Address.Query" button to send the information into SSC.

You should receive a list of all of the Address Codes in SunSystems.

When you are happy with the results, click on the browsers "back" button until you are back on the main screen.

EXERCISE 2 – Add a new Address record

Select the “Address” component, and then click on “CreateOrAmend”. Then from the list which appears, select “Execute”.

Enter “PKP” in the Business Unit field in the Header section of the screen.

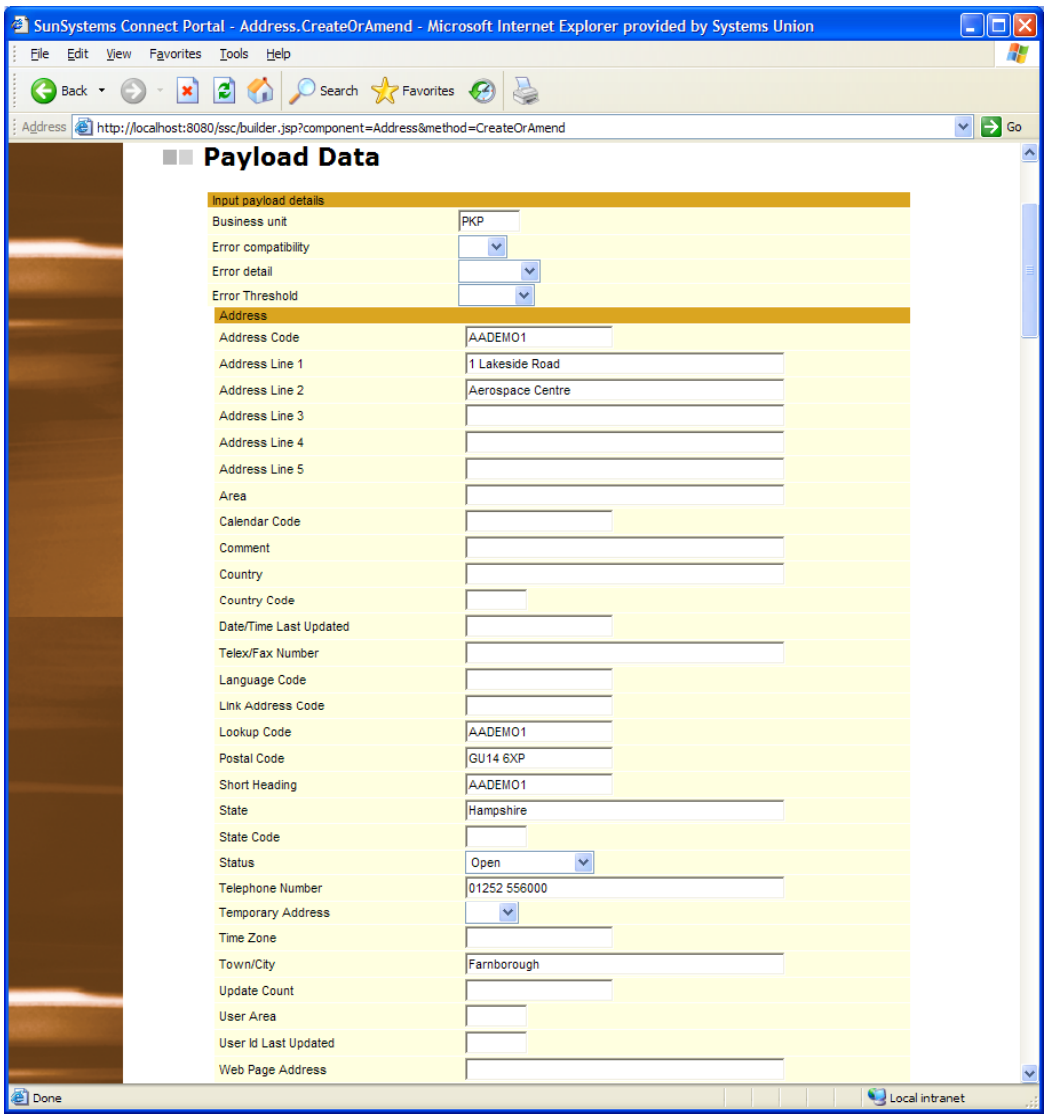
Go down to the Details section, and under Address Code, enter the value “AADEMO1”.

Fill in Address Lines 1 and 2 with something, and also Town, State, and Post Code.

Select a Status of “Open”.

Under Lookup Code and Short Heading, enter “AADEMO1”.

Enter a telephone number too. Your screen should now look something like this:

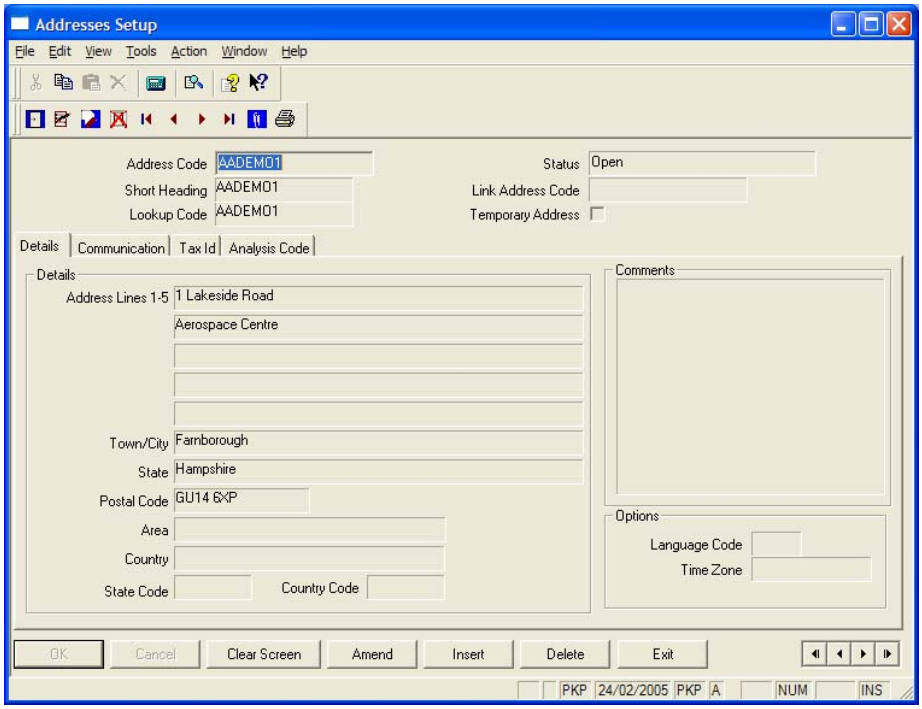


Click on “Execute Address.CreateOrAmend”. After a short while, you should receive a confirmation message of success after the record has been updated to SunSystems:


```
  <?xml version="1.0" encoding="UTF-8" ?>
- <SSC>
  - <User>
      <Name>PKP</Name>
    </User>
  - <SunSystemsContext>
      <BusinessUnit>PKP</BusinessUnit>
    </SunSystemsContext>
  - <Payload>
    - <Address status="success">
        <AddressCode>AADEMO1</AddressCode>
      </Address>
    </Payload>
  </SSC>
```


This is SSC’s own version of XML, called SSML. Do not worry if this doesn’t make sense just yet, SSML is discussed fully in the next chapter.

Check that the record exists by using the Address Setup (AS) function within SunSystems and you should see the record as shown:



Back on the Web Page screen, click on the browsers “back” button until you are back on the main screen.

EXERCISE 3 – Query a selection of ChartOfAccounts data

Select the “Accounts” component, and then click on “Query”. Then from the list which appears, select “Execute”.

In the Header section of the resulting screen, enter “PKP” in the Business Unit field.

In order to make sure that only the records we want are retrieved, we need to add a filter to the request.

Enter the following text into the Filter field:

(/Accounts/Account Code <> 64001,64005) OR (/Accounts/Account Code = 81005)

Note that where you use AND or OR conditions, you must surround the criteria with parenthesis.

Select the tickbox next to “Accounts”. This means that the query will retrieve everything within the Accounts payload structure, unlike in exercise 1, where we just selected the Address Code.

Accounts ☑

Again, you can view the SSML to be sent into SSC by clicking on the View Payload button:

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <SSC>
    <ErrorContext />
    <User />
  - <SunSystemsContext>
      <BusinessUnit>PKP</BusinessUnit>
    </SunSystemsContext>
  - <Payload>
    - <Filter>
      - <Expr operator="OR">
          <Item name="/Accounts/AccountCode" operator="BETWEEN" minvalue="64001" maxvalue="64005" />
          <Item name="/Accounts/AccountCode" operator="EQU" value="81005" />
        </Expr>
      </Filter>
    - <Select>
        <Accounts>*</Accounts>
      </Select>
    </Payload>
  </SSC>
```

Here you can see the difference between the Web page notation for formatting filter statements, and the SSML notation.

Click “back”, and then “Execute Accounts.Query” to run the Query.

You should see the results of the query, with the filter criteria as specified above…

Click on the browsers “back” button until you are back on the main screen.

EXERCISE 4 – View the Schema for Sales Order CreateOrAmend Input payloads

Select the “SalesOrder” component (listed under “SunSystems 5 Order Fulfilment”), and then click on “CreateOrAmend”. Then from the list which appears, select “View input XML Schema”.

You will be presented with the XML Schema, which defines the SSML payloads going in to the Sales Order CreateOrAmend method.

Chapter 4 – Connectivity

SOAP

SOAP is an XML based standard for interprocess messaging and execution. It is rapidly becoming the most important technique for building web services and enterprise-wide integrations. In particular, Microsoft .NET is built around SOAP, as are the major integration tools such as BizTalk and WebSphere.

SOAP Advantages

- It is a well-known standard; documentation is readily available, and mature libraries can be used.
- It is multi-platform. If you are not using Java, SOAP may be the only choice. This may also be true when using external integration tools such as Microsoft BizTalk or IBM WebSphere.
- It is passed over HTTP connections, using XML formatting. This is well understood by firewalls, and their administrators, making deployment on connected systems easy.

The end-point URL for all SOAP communication with SSC is:

http://*<SSC_Server>*:*<port>*/ssc/servlet/SUGSoap

WSDL

WSDL (Web Services Description Language) is an information service that describes available web services. Connect provides dynamically generated WSDL services that describe the SOAP interfaces for all components.

Several methods of interacting with WSDL exist. These are primarily for use with versions of SunSystems PRIOR to v5.2.1, however they can still be used if required.

The standard WSDL document is accessed with http://*<SSC_Server>*:*<Port>*/ssc/wsdl.jsp. The WSDL document generated by this service lists each Connect component individually, which produces a very large document, and can be slow to parse. This is not generally a problem for clients that use this information as a one-off hit during development, but it can cause performance problems for libraries that refer to the WSDL information every time they run (e.g. the Microsoft SOAP Toolkit).

To avoid these problems an alternate SOAP entry point is provided, which is described by http://*<SSC_Server>*:*<Port>*/ssc/execwsdl.jsp. This initialises much quicker than the previous document and provides access to all components, as the component and method you wish to call are specified in the call.

These WSDL interfaces can be referenced from most, if not all development environments including Visual Studio 6, Visual Studio .NET (as Web References) and Java.

Web References in .NET

From SunSystems 5.2.1 another WSDL interface was exposed. Designed for use within the .NET environment, you can also use the following WSDL document and use it as a Web Reference.

http://*<SSC_Server>*:*<port>*/ssc/comp-wsdl.jsp?component=*<Component_Name>*

It exposes each component as a separate Web Reference and provides more efficient functionality than its 5.1.5 predecessors…

XML

XML stands for Extensible Markup Language. It was designed by W3C to be a universal format for encoding information in a structured manner. XML deals with a documents structure rather than its presentation.

The data passed between 3rd party client applications and the SSC interface is in XML format. This XML adheres to various schema definitions laid down by SunSystems which describe the data. XML schema definitions exist for both the input and output payloads for every method of every component.

Collectively, the XML which is formatted according to these schema definitions is known as SSML. This stands for SunSystems Markup Language and simply refers to the XML which is generated according to the SSC schema definitions.

Special XML Characters

In all XML, there are certain characters which have special meanings within an XML document. Therefore if you are wish to use any of the following please ensure you use the appropriate specified control codes according to the following table:

"' must be specified as '	"" must be specified as "
'&' must be specified as &	'£' must be specified as £
'¬' must be specified as ¬	'<' must be specified as <
'>' must be specified as >	

SSML

An SSML payload file is simply a text document in XML format. A payload contains the XML elements and data values used by SSC at runtime to import, update or export data from the SunSystems database.

We have already seen a glimpse of SSML in the previous chapter on the SSC Demo Web Page.

Each SSC component has a payload definition controlling the data items that you can specify for that particular component. All components have a set of one or more methods that SSC uses at runtime to execute, such as Create, Amend or Query.

Input Payloads

An input payload contains the XML elements and their data values that SSC processes at runtime. Input payloads generally have a common structure. However, those used by the Query and Process methods are different.

A highly generalised structure of an SSC input XML payload file is shown below:

```
<SSC>
    <ErrorContext>
        <CompatibilityMode></CompatibilityMode>
        <ErrorOutput></ErrorOutput>
        <ErrorThreshold></ErrorThreshold>
    </ErrorContext>
    <User>
        <Name></Name>
    </User>
    <SunSystemsContext>
        <BusinessUnit></BusinessUnit>
    </SunSystemsContext>
    <MethodContext>
        <Group>
            <Tag1>
            <Tag2>
        </Group>
    </MethodContext>
    <Payload>
        <ComponentName>
            <Tag1></Tag1>
            <Tag2></Tag2>
            <Section1>
                <Section1Tag1></Section1Tag1>
                <Section1Tag2></Section1Tag2>
            </Section1>
            <Section2>
                <Section2Tag1></Section2Tag1>
                <Section2Tag2></Section2Tag2>
            </Section2>
        </ComponentName>
    </Payload>
</SSC>
```

The input payload is divided into five sections:

- <ErrorContext> is an optional node within the XML. If specified it overrides the default settings specified within SunSystems. These will be covered in more detail later in the course.
- <User> identifies the SunSystems User ID that is to be used to execute the payload. This must be a user already created in SunSystems.
- <SunSystemsContext> identifies the context in which the payload is to be run, such as the business unit, budget or ledger code.
- <MethodContext> identifies component specific context parameters. Not always used but important for those components which do utilise it. E.g. for Journal this is where you specify default account codes, posting options, balancing options, journal types and other run-time options.

- <Payload> contains the identity of the component that is to be executed, together with the tags and tag values for data that are used to identify the records that are to be processed and the values for data items to be created or amended. Depending on the component, there may be dependent or related data and these are conveniently organized into sections.

Each component has an associated payload definition that is defined by SunSystems containing all the tags that can be specified. SSC is installed with a variety of online tools that allow users to view and work with the payload definitions, as well as creating payload files.

Payloads can be created using any text editor, such as Windows Notepad or Wordpad. Depending on user expertise, a dedicated XML editor may be easier and more appropriate to use, however in most SSC solutions the XML will be built up dynamically within an application before being sent to the SSC Server.

Output Payloads

At runtime, SSC processes the input payload file and generates an output payload file (also in XML format), that records the success or failure of the records contained in the input file and the data item values that were written to or retrieved from database during execution.

A highly generalised structure of an SSC output XML payload file is shown below:

```
<SSC>
        <User>
                <Name></Name>
        </User>
        <SunSystemsContext>
                <BusinessUnit></BusinessUnit>
        </SunSystemsContext>
        <Payload>
                <ComponentName status="success">
                        <PrimaryKey></PrimaryKey>
                </ComponentName>
                <ComponentName status="success">
                        <PrimaryKey></PrimaryKey>
                </ComponentName>
        </Payload>
</SSC>
```

The outputpayload is generally divided into three sections:

- <User> identifies the SunSystems User ID that was used to execute the payload.
- <SunSystemsContext> identifies the context in which the payload was run, such as the business unit, budget or ledger code.
- <Payload> contains information about wether or not the import was successful or not.

For each record that was imported the Primary Key is returned along with a status attribute of ‘success’ or ‘fail’.

Failures will be returned with all the fields specified in the input payload, along with the appropriate error messages from SSC and/or the SunSystems back-end.

SSML differences between Imports and Queries

Where you are importing data, it is important to specify the data you wish to enter into SunSystems. For example:

```
<SSC>
        <User>
                <Name>PKP</Name>
        </User>
        <SunSystemsContext>
                <BusinessUnit>PKP</BusinessUnit>
        </SunSystemsContext>
        <Payload>
                <Address>
                        <AddressCode>DemoAddress</AddressCode>
                        <AddressLine1>AddressLine1</AddressLine1>
                        <TownCity>My Town</TownCity>
                        <State>My State</State>
                        <PostalCode>GU14 6XP</PostalCode>
                        <TelephoneNumber>01234 567890</TelephoneNumber>
                </Address>
        </Payload>
</SSC>
```

This would create an address record, giving it a few pieces of information associated with it such as town, city, and telephone number etc…

However if you wish to *query* an address record, you need completely different information passed through in the <Payload> tags of the above XML.

The following XML examples are from the <Payload> level down only, and assume it is encapsulated within the root <SSC> tags.

As a minimum, the following is required for a query:

```
        <Payload>
                <ComponentName>*</ComponentName>
        </Payload>
```

This states that you want to retrieve all address information from within SunSystems, however this is often not a good idea – from both a functional and performance perspective.

You can restrict the information you wish to be returned in 2 ways:-

- Specify a FILTER.
- Specify the FIELDS you wish to be returned.

The reasoning behind this is that firstly, applying filter criteria means that you only retrieve the records you want. This means less work by SSC, less work at the database level, and less data transmitted back over the network. It also means a smaller amount of data to be read in by your application, speeding that up too.

A filter on its own will narrow down the records being returned but still return ALL data for that record. Specifying the fields you want allows you to further restrict the data being returned.

For example, if you only wish to retrieve basic address information, specifying those fields in the input payload of a query may help prevent unnecessary SQL queries being run and views to be accessed – again speeding up the overall processing time, as well as reducing the size of the payload being returned back across the network. This is especially useful where a potentially large range of records is being queried.

We will now look at how to format both these filters and field selections.

Filters

The general format of a filter in an input payload is as follows:

```
<Payload>
        <Filter>
                <Item name="/ComponentName/TagToQueryBy" operator="EQU"
                value="SearchCriteria" />
        </Filter>
</Payload>
```

You can specify multiple conditions by expanding the above as follows:

```
<Payload>
        <Filter>
                <Expr operator="AND">
                        <Item name="/ComponentName/TagToQueryBy"
                        operator="EQU" value="SearchCriteria" />
                        <Item name="/ComponentName/TagToQueryBy2"
                        operator="EQU" value="SearchCriteria2" />
                </Expr>
        </Filter>
</Payload>
```

Valid values for the <Expr> operator attribute are:

- AND
- OR

The filter operators are slightly different when specified directly within SSML, than when entered on the web page (previous chapter). Here is a table describing the different terms.

Meaning	Web Page Operator	SSML Operator
Equals	=	EQU
Not Equal	!=	NEQU
Greater Than	>	GT
Greater Than Or Equals	>=	GTE
Less Than	<	LT
Less Than Or Equals	<=	LTE
List	><	IN
Like	~=	LIKE
Range	<>	BETWEEN

Selecting specific fields

The general format of a select in an input payload is as follows:

```
<Payload>
    <Select>
        <ComponentName>*</ComponentName>
    </Select>
</Payload>
```

In the example above, ALL information for the given component would be returned, as the asterisk (*) denotes that you wish to retrieve this node and all its children. To specify only a few fields, specify the individual tags you wish to have returned.

For example:

```
<Payload>
    <Select>
        <Address>
            <AddressCode />
            <AddressLine1 />
            <AddressLine2 />
            <AddressLine3 />
            <TownCity />
            <State />
            <PostalCode />
            <Status />
            <TelephoneNumber />
        </Address>
    </Select>
</Payload>
```

Will retrieve only the main fields required when querying address information.

Putting it together

Here is an example of an input payload which might be submitted to the Address component and Query method:

```
<?xml version="1.0" encoding="UTF-8"?>
<SSC>
    <User>
        <Name>PKP</Name>
    </User>
    <SunSystemsContext>
        <BusinessUnit>PKP</BusinessUnit>
    </SunSystemsContext>
    <Payload>
        <Filter>
            <Expr operator="OR">
                <Item name="/Address/AddressCode" operator="EQU" value="64001"/>
                <Item name="/Address/AddressCode" operator="EQU" value="64002"/>
            </Expr>
        </Filter>
        <Select>
            <Address>
                <AddressCode/>
                <TelephoneNumber/>
            </Address>
        </Select>
    </Payload>
</SSC>
```

This example will return just the address code and telephone number for the addresses ‘64001’ and ‘64002’.

Related Data

It is possible with some components to include related data within the main <Payload>\<*Component*> body.

An example is below:

```
<?xml version="1.0" encoding="UTF-8"?>
<SSC>
    <User>
        <Name>PKP</Name>
    </User>
    <SunSystemsContext>
        <BusinessUnit>PKP</BusinessUnit>
    </SunSystemsContext>
    <Payload>
        <Supplier>
            <AccountCode>SSCDEMO</AccountCode>
            <Description>This is a demonstration</Description>
            <LookupCode>SSCDEMO</LookupCode>
            <ShortHeading>SSCDEMO</ShortHeading>
            <SupplierCode>SSCDEMO</SupplierCode>
            <Accounts>
                <AccountCode>SSCDEMO</AccountCode>
                <AccountType>1</AccountType>
                <BalanceType>1</BalanceType>
            </Accounts>
        </Supplier>
    </Payload>
</SSC>
```

This example, using the Supplier/CreateOrAmend method, creates a Supplier called “SSCDEMO”. It also creates a Chart Of Account record called “SSCDEMO” and links the two together (using the <AccountCode> tag in the Supplier node).

In this case, the Accounts data is referred to as “related” data.

To see what related data can be attached for given components and methods, view an empty input schema, and you will see the various child nodes displayed.

> **Note:** If using DJI methods, there may be occasions where it does not accept some related data. If you are using the DJI methods, it is recommended you run a check through the demonstration web page (discussed later) to see whether it accepts the related data or not.

If querying data, and electing to return all the data for a given record, related information, should any exist, will also be returned. You can also choose to bring the data back from within your <Select> tags, thus:

```
    <Select>
        </Supplier>
            <AccountCode />
            <SupplierCode />
            <Accounts>
                <AccountCode />
                <AccountType />
            </Accounts>
        </Supplier>
    </Select>
```

Note: It is also advisable to use the related data functionality only to create the related data items. On-going maintenance of the data items should be done using that data items individual SSC component.

Client Libraries

Access to the SSC functionality is primarily exposed through the use of the SOAP interface as described above. However, provided with SunSystems is a set of Client Libraries. Two for Java, one for older 32-bit windows applications (such as C++ / VB6 apps), and one specifically for .NET languages.

These client libraries provide access to SSC as well as other useful functionality. The first, "clientlib.jar", can be used as a gateway to SSC from other Java applications. This resides in the <SunSystems5>\ssc\lib directory. clientlib.jar is a redistributable package so you can ship it with your Java-based SSC enabled applications to ensure you use the correct version. An alternative version – "connect-client.jar" is also available which includes support for multiple versions of the JDK.

The next Client Library is called "SSCClientLibrary.dll". This is the one for use with VB6 or older win32 programming languages. It resides in the <SunSystems5>\ssc\bin directory and has a dependency – sockets.dll. The sockets.dll file resides in the base <SunSystems5> directory.

Finally, the third Client Library is the "SunSystemsConnectClient.dll", and is for use by Microsoft Visual Studio .NET. Again, this resides in the base <SunSystems5> directory. Its dependencies are the win32 dll's – SSCClientLibrary.dll and sockets.dll, as it is essentially a .NET wrapper for the aforementioned win32 library.

All of these will be fully discussed and demonstrated in the Coding section.

Security Manager

SSC is an open interface, it is designed to be flexible, so that it can communicate with a wide range of applications.

This however gives rise to the possibility of abuse, which means that people who are not authorised users of the system might try to gain access to the sensitive information stored within SunSystems.

In versions of SunSystems prior to 5.2.1, the solution to this security issue was to have a screen within SunSystems where administrators could specify individual SunSystems operators who they wished to be able to access SSC. When a payload was submitted, the user ID specified in the <Name> tag under the <User> node was validated against the list held in SunSystems. If there was a match, the payload was allowed to be processed. A limitation of this solution is that once a 'valid' user had been identified, anyone could put that user ID into their input payload and access SSC.

In SunSystems 5.2.1 this was improved upon by the introduction of security manager. This is a system whereby the client requests a digitally signed "voucher" string, which must then be passed through along with the input payloads in all subsequent calls to SSC.

The voucher is an encrypted string of text, containing user information, permissions information, and a digital signature, as well as current session information. This means that any given voucher is only valid for the lifetime of the current process running it. Because of this, you cannot save the returned voucher string and re-use it for subsequent runs of the

program; it is only valid for the current run (although you can use it as many times as you want until the process finishes).

The digital signature prevents forged vouchers from being created.

A security voucher looks like this example here:


```
"ssvoucher:interactive|session:z7i0/P8z|user:MAT|grant:ssc|grant:td|grant:ad
|sig:SV=Wl6FBY6nhtxlog+8ev5RY9BA6AE40ZFKxEys/9F5uzxo1UqU30VSXQ/aXbRq/ONTj3qQ
ps1gkQFk+U+sx2Zynw=="
```


In order to retrieve a voucher, you must connect to security manager using the interfaces exposed in the client libraries included with SunSystems. Security manager requires a valid SunSystems user ID and password, which is then submitted securely for validation by the SSC Server using proprietary encryption and communication. If correct, a status of AUTH_SUCCESS is returned and the client can then query security manager for the voucher. It is the responsibility of the client application to handle incorrect passwords in the manner as determined by their own business logic.

When connecting using the win32 client library you need to specify a port number. If you are using the Java or .NET client libraries this is optional, as it defaults. However you may still choose to specify it anyway. This port number changes between SunSystems 5.2.1 and 5.2.1 SP1. Ensure you use the right port, otherwise the call to security manager will fail with a ‘socket error 2’.

This table specifies the different port numbers:

Version	Port
5.2.1	50550
5.2.1 SP1	55000

Use of security manager is fully demonstrated in Chapter 5 of this manual and additional documentation of the DLL methods for Win32 calls is available in “Appendix B: Security Manager DLL methods”.

Chapter 5 – Client side coding for SSC

This section is designed to demonstrate the use of SSC in as many ways as is practically possible within the confines of this manual.

This section is split up by language, so it should make for easy reference. It also demonstrates different ways of calling SSC within each language where applicable.

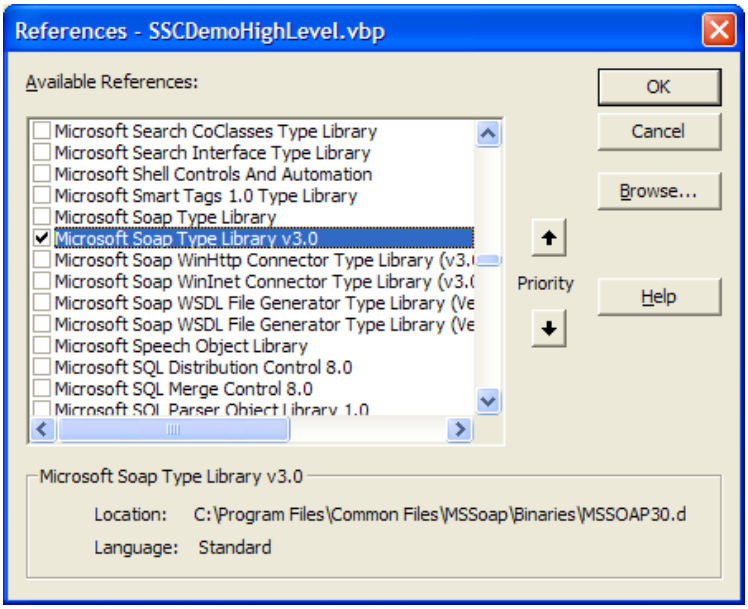
Every program shown has the same functionality:

- Creates an input payload for address code "Demo1" with basic address information.
- Calls SSC Address component / CreateOrAmend method passing it the input payload.
- Where security manager is used, use PKP user ID and blank password to retrieve voucher string.
- Includes basic error checking
- Returns the resulting output XML payload to the screen.
- The form contains 3 controls:
    - Text box for the target SSC Server Name
    - Text box for results
    - "Go" button

General Rules

For Win32 applications the installation of a SOAP Toolkit such as Microsoft SOAP Toolkit 3.0 must be installed to facilitate the client initiation of a SOAP call.

For all Visual Studio 6 languages, you must make a reference to the SOAP Toolkit you wish to utilise. In the case of the Microsoft SOAP Toolkit, do this by selecting "References" from the "Project Menu". Then scroll down until you find "Microsoft Soap Type Library v3.0". Place a tick next to it and click OK.

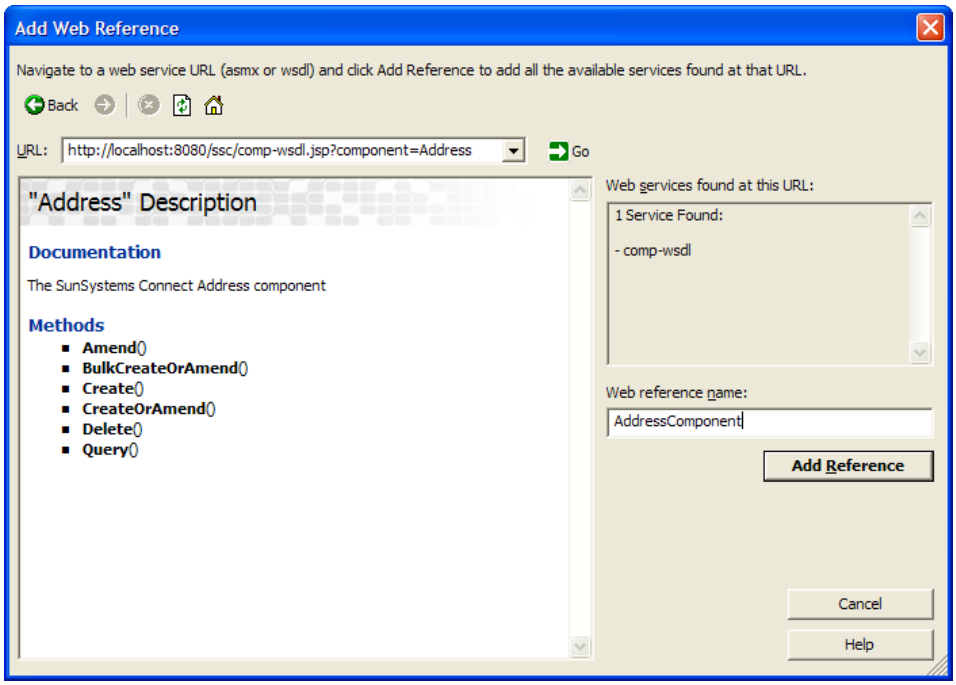


Additionally, for access to Security Manager you need to copy the following 2 files to your project folder:

- <SunSystems5>\sockets.dll
- <SunSystems5>\ssc\bin\SSCClientLibrary.dll

For each of the .NET Demo's we will be using a Web Reference to communicate with SSC. Create it by following these steps:

1. Select "Add Web Reference" from the Project menu.
2. Enter the URL of the Connect WSDL service into the URL field, e.g. http://SSCServer:8080/ssc/comp-wsdl.jsp?component=Address
3. Press Go.
4. After a short pause, the methods of the Address component should be displayed:

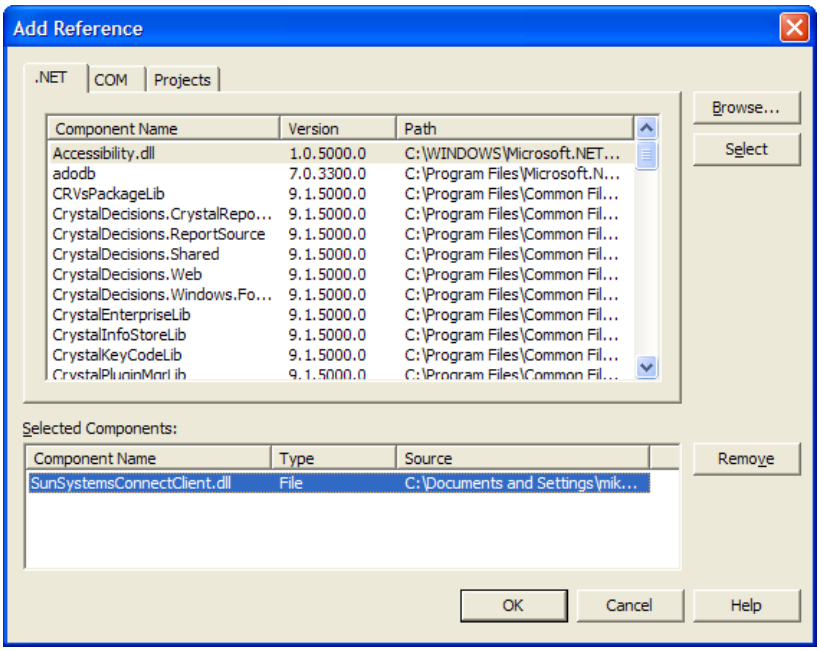


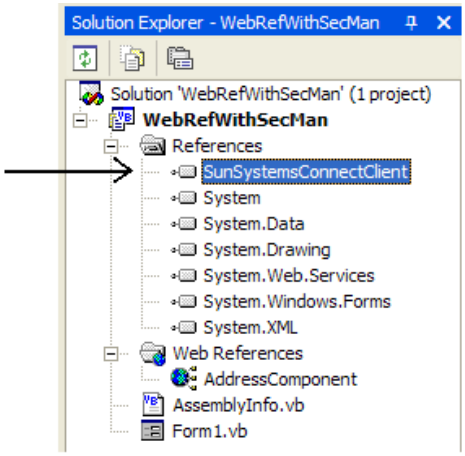
5. Change the name of the reference to "AddressComponent" as shown above.
6. Press "Add Reference".

In order to use security manager in .NET, copy the following 3 files to your project\bin folder:

- <SunSystems5>\sockets.dll
- <SunSystems5>\ssc\bin\SSCClientLibrary.dll
- <SunSystems5>\SunSystemsConnectClient.dll

Create a reference to the SunSystemsConnectClient.dll by selecting “Add Reference” from the Project Menu. In the dialog box that follows, ensure you are on the .NET tab and click “Browse…”. From here navigate to your copy of the SunSystemsConnectClient .dll and select it. Click “open”, you will see it listed as below. Click OK to add it as a reference as shown below:





Note: If desired, you can also use the Microsoft SOAP Toolkit 3.0 from .NET. Just create a reference to it in the same way as above, except that the “Microsoft SOAP Type Library v3.0” is already listed and can be found under the COM tab on the Add Reference dialog screen.

Important note: The following code source is for demonstration purposes only and should be used only as a guide. It should not be assumed to be the de-facto way of calling SSC, as every programmer has their own unique way of coding.

Visual Basic 6

High Level WSDL call – no security manager

This goes in the main form code:

```
Private Sub btnGo_Click()

  On Error Resume Next

  'Create instance of SOAP client
  Dim soapClient
  Set soapClient = CreateObject("MSSOAP.SoapClient30")
  Call soapClient.MSSoapInit("http://" & Text1.Text & ":8080/ssc/execwsdl.jsp",
"", "")

  'Build input payload
  Dim sInputPayload As String
  sInputPayload = "<?xml version=""1.0"" encoding=""UTF-8"" ?>" & _
                  "<SSC>" & _
                  "  <User>" & _
                  "    <Name>PKP</Name>" & _
                  "  </User>" & _
                  "  <SunSystemsContext>" & _
                  "    <BusinessUnit>PKP</BusinessUnit>" & _
                  "  </SunSystemsContext>" & _
                  "  <Payload>" & _
                  "    <Address>" & _
                  "      <AddressCode>Demo1</AddressCode>" & _
                  "      <AddressLine1>My Street</AddressLine1>" & _
                  "      <TownCity>My Town</TownCity>" & _
                  "      <PostalCode>AB12 3CD</PostalCode>" & _
                  "      <TelephoneNumber>01234 567890</TelephoneNumber>" & _
                  "      <LookupCode>DEMO1</LookupCode>" & _
                  "    </Address>" & _
                  "  </Payload>" & _
                  "</SSC>"

  'Send payload into SSC
  Dim sOutputPayload As String
  sOutputPayload = soapClient.run("Address", "CreateOrAmend", sInputPayload)

  'Check for errors
  If Err.Number <> 0 Then
    Text2.Text = Err.Description & vbCrLf
    Text2.Text = Text2.Text & " faultcode=" & soapClient.FaultCode & vbCrLf
    Text2.Text = Text2.Text & " faultstring=" & soapClient.FaultString & vbCrLf
    Text2.Text = Text2.Text & " faultactor=" & soapClient.FaultActor & vbCrLf
    Text2.Text = Text2.Text & " detail=" & soapClient.Detail
  Else
    'No fatal errors in call, so put returned payload into textbox
    Text2.Text = sOutputPayload
  End If

End Sub
```


High Level WSDL call – with security manager

This goes in the main form code:

```
Private Sub btnGo_Click()

  On Error Resume Next

  'Get Security Voucher
  Dim AuthoriseResult As Integer
  Dim sSSCVoucher As String

  AuthoriseResult = SSCAuthorise(Text1.Text, 55000, "PKP", "")

  If Not AuthoriseResult = 0 Then
    Text2.Text = "An error occurred in Validation: " & SSCGetErrorResponse
    Exit Sub
  End If

  sSSCVoucher = SSCReadVoucher

  'Create instance of SOAP client
  Dim soapClient
  Set soapClient = CreateObject("MSSOAP.SoapClient30")
  Call soapClient.MSSoapInit("http://" & Text1.Text & ":8080/ssc/execwsdl.jsp",
"", "")

  'Build input payload
  Dim sInputPayload As String
  sInputPayload = "<?xml version=""1.0"" encoding=""UTF-8"" ?>" & _
                  "<SSC>" & _
                  "  <User>" & _
                  "    <Name>PKP</Name>" & _
                  "  </User>" & _
                  "  <SunSystemsContext>" & _
                  "    <BusinessUnit>PKP</BusinessUnit>" & _
                  "  </SunSystemsContext>" & _
                  "  <Payload>" & _
                  "    <Address>" & _
                  "      <AddressCode>Demo1</AddressCode>" & _
                  "      <AddressLine1>My Street</AddressLine1>" & _
                  "      <TownCity>My Town</TownCity>" & _
                  "      <PostalCode>AB12 3CD</PostalCode>" & _
                  "      <TelephoneNumber>01234 567890</TelephoneNumber>" & _
                  "      <LookupCode>DEMO1</LookupCode>" & _
                  "    </Address>" & _
                  "  </Payload>" & _
                  "</SSC>"

  'Send payload into SSC
  Dim sOutputPayload As String
  sOutputPayload = soapClient.runAuth("Address", "CreateOrAmend", _
                                      sSSCVoucher, sInputPayload)

  'Check for errors
  If Err.Number <> 0 Then
    Text2.Text = Err.Description & vbCrLf
    Text2.Text = Text2.Text & " faultcode=" & soapClient.FaultCode & vbCrLf
    Text2.Text = Text2.Text & " faultstring=" & soapClient.FaultString & vbCrLf
    Text2.Text = Text2.Text & " faultactor=" & soapClient.FaultActor & vbCrLf
    Text2.Text = Text2.Text & " detail=" & soapClient.Detail
  Else
    'No fatal errors in call, so put returned payload into textbox
    Text2.Text = sOutputPayload
  End If

End Sub
```

This goes in a module:

```
'Declare SSC Client Library Functions

Public Declare Function SSCAuthorise Lib "SSCClientLibrary.dll" _
Alias "_SSCAuthorise@16" (ByVal HostName As String, ByVal Port As Long, _
ByVal UserName As String, ByVal Password As String) As Integer

Public Declare Function SSCReadVoucher Lib "SSCClientLibrary.dll" _
Alias "_SSCReadVoucher@0" () As String

Public Declare Function SSCGetErrorResponse Lib "SSCClientLibrary.dll" _
Alias "_SSCGetErrorRespose@0" () As String

Public Declare Function SSCGetExtendedErrorCode Lib "SSCClientLibrary.dll" _
Alias "_SSCGetExtendedErrorCode@0" () As String

Public Declare Function SSCGetExtendedErrorMessage Lib "SSCClientLibrary.dll" _
Alias "_SSCGetExtendedErrorMessage@0" () As String
```

Low Level SOAP call – no security manager

This goes in the main form code:

```
Private Sub btnGo_Click()

  On Error Resume Next

  'Build input payload
  Dim sInputPayload As String
  sInputPayload = "<?xml version=""1.0"" encoding=""UTF-8"" ?>" & _
                  "<SSC>" & _
                  "  <User>" & _
                  "    <Name>PKP</Name>" & _
                  "  </User>" & _
                  "  <SunSystemsContext>" & _
                  "    <BusinessUnit>PKP</BusinessUnit>" & _
                  "  </SunSystemsContext>" & _
                  "  <Payload>" & _
                  "    <Address>" & _
                  "      <AddressCode>Demo1</AddressCode>" & _
                  "      <AddressLine1>My Street</AddressLine1>" & _
                  "      <TownCity>My Town</TownCity>" & _
                  "      <PostalCode>AB12 3CD</PostalCode>" & _
                  "      <TelephoneNumber>01234 567890</TelephoneNumber>" & _
                  "      <LookupCode>DEMO1</LookupCode>" & _
                  "    </Address>" & _
                  "  </Payload>" & _
                  "</SSC>"

  'Call function to send payload into SSC
  Dim sOutputPayload As String
  sOutputPayload = InvokeSOAPCall("Address", "CreateOrAmend", sInputPayload)

  'Put returned text into Textbox - error checking occurs in module so this
  'could be valid XML or it could be some error text.
  Text2.Text = sOutputPayload

End Sub
```

This code goes in a module:

```
Public Function InvokeSOAPCall(strComponent As String, strMethod As String,
                               strPayload As String) As String

    On Error Resume Next

    Dim strResult

    ' Declare and initialize the Soap Objects Required to Connect
    ' to the server, build the SOAP Message, and Read the SOAP
    ' Response Message
```



```
    Dim Connector
    Dim Serializer
    Dim Reader

    Set Connector = CreateObject("MSSOAP.HttpConnector30")
    Set Serializer = CreateObject("MSSOAP.SoapSerializer30")
    Set Reader = CreateObject("MSSOAP.SoapReader30")

    'Wrap payload going into SSC with CDATA tags
    strPayload = "<![CDATA[" & strPayload & "]]>"

    ' The following Connector settings establish the SOAP Listener connection
    Connector.Property("EndPointURL") = "http://" & frmMain!Text1.Text & _
                                        ":8080/ssc/servlet/SUGSoap"
    Connector.Property("SoapAction") = "uri:SystemsUnion-" & strComponent
    Connector.Property("Timeout") = "3600000"
    Connector.Connect

    ' The following builds the SOAP Message by Using the
    ' Serializer which generates the XML SOAP Message
    ' and executes the SOAP Request
    Connector.BeginMessage
        Serializer.Init Connector.InputStream
        Serializer.StartEnvelope
        Serializer.SoapNamespace "xsi", "http://www.w3.org/1999/XMLSchema-
instance"
        Serializer.StartBody
        Serializer.StartElement strMethod, "urn:SystemsUnion-" & strComponent,
, "m"
        Serializer.StartElement "strPayload"
        Serializer.WriteXml strPayload
        Serializer.EndElement
        Serializer.EndElement
        Serializer.EndBody
        'This line actually fires off the call to SSC.
        Serializer.EndEnvelope
    Connector.EndMessage

    'Check for TimeOut
    If Err.Number = 5400 Then
      InvokeSOAPCall = "Connection timed out"
      Connector.Reset
      Exit Function
    End If


    ' The following Reader object opens the SOAP Response
    ' and evaluates if any SOAP Fault was raised
    ' The SOAP Response is placed in the strResult variable
    Reader.Load Connector.OutputStream
    If Not Reader.Fault Is Nothing Then
        strResult = Reader.FaultString.Text
    Else
        strResult = Reader.RpcResult.Text
    End If

    'Reset the SOAP Connection
    Connector.Reset

    InvokeSOAPCall = strResult

End Function
```

Low Level SOAP call – with security manager

This goes in the main form code:

```
Private Sub btnGo_Click()

  On Error Resume Next

  'Get Security Voucher
  Dim AuthoriseResult As Integer
  Dim sSSCVoucher As String

  AuthoriseResult = SSCAuthorise(Text1.Text, 55000, "PKP", "")

  If Not AuthoriseResult = 0 Then
    Text2.Text = "An error occurred in Validation: " & SSCGetErrorResponse
    Exit Sub
  End If

  sSSCVoucher = SSCReadVoucher

  'Build input payload
  Dim sInputPayload As String
  sInputPayload = "<?xml version=""1.0"" encoding=""UTF-8"" ?>" & _
                  "<SSC>" & _
                  "  <User>" & _
                  "    <Name>PKP</Name>" & _
                  "  </User>" & _
                  "  <SunSystemsContext>" & _
                  "    <BusinessUnit>PKP</BusinessUnit>" & _
                  "  </SunSystemsContext>" & _
                  "  <Payload>" & _
                  "    <Address>" & _
                  "      <AddressCode>Demo1</AddressCode>" & _
                  "      <AddressLine1>My Street</AddressLine1>" & _
                  "      <TownCity>My Town</TownCity>" & _
                  "      <PostalCode>AB12 3CD</PostalCode>" & _
                  "      <TelephoneNumber>01234 567890</TelephoneNumber>" & _
                  "      <LookupCode>DEMO1</LookupCode>" & _
                  "    </Address>" & _
                  "  </Payload>" & _
                  "</SSC>"

  'Call function to send payload into SSC
  Dim sOutputPayload As String
  sOutputPayload = InvokeSOAPCall("Address", "CreateOrAmend", sSSCVoucher, _
                                  sInputPayload)

  'Put returned text into Textbox - error checking occurs in module so this
  'could be valid XML or it could be some error text.
  Text2.Text = sOutputPayload

End Sub
```

This goes in a module:

```
'Declare SSC Client Library Functions

Public Declare Function SSCAuthorise Lib "SSCClientLibrary.dll" _
Alias "_SSCAuthorise@16" (ByVal HostName As String, ByVal Port As Long, _
ByVal UserName As String, ByVal Password As String) As Integer

Public Declare Function SSCReadVoucher Lib "SSCClientLibrary.dll" _
Alias "_SSCReadVoucher@0" () As String

Public Declare Function SSCGetErrorResponse Lib "SSCClientLibrary.dll" _
Alias "_SSCGetErrorRespose@0" () As String

Public Declare Function SSCGetExtendedErrorCode Lib "SSCClientLibrary.dll" _
Alias "_SSCGetExtendedErrorCode@0" () As String

Public Declare Function SSCGetExtendedErrorMessage Lib "SSCClientLibrary.dll" _
Alias "_SSCGetExtendedErrorMessage@0" () As String
```



```
Public Function InvokeSOAPCall(strComponent As String, strMethod As String,
strSSCVoucher As String, strPayload As String) As String

    On Error Resume Next

    Dim strResult

    ' Declare and initialize the Soap Objects Required to Connect
    ' to the server, build the SOAP Message, and Read the SOAP Response Message
    Dim Connector
    Dim Serializer
    Dim Reader

    Set Connector = CreateObject("MSSOAP.HttpConnector30")
    Set Serializer = CreateObject("MSSOAP.SoapSerializer30")
    Set Reader = CreateObject("MSSOAP.SoapReader30")

    'Wrap payload going into SSC with CDATA tags
    strPayload = "<![CDATA[" & strPayload & "]]>"

    ' The following Connector settings establish the SOAP Listener connection
    Connector.Property("EndPointURL") = "http://" & frmMain!Text1.Text & _
                                        ":8080/ssc/servlet/SUGSoap"
    Connector.Property("SoapAction") = "uri:SystemsUnion-" & strComponent
    Connector.Property("Timeout") = "3600000"
    Connector.Connect

    ' The following builds the SOAP Message by Using the
    ' Serializer which generates the XML SOAP Message
    ' and executes the SOAP Request
    Connector.BeginMessage
        Serializer.Init Connector.InputStream
        Serializer.StartEnvelope
        Serializer.SoapNamespace "xsi", "http://www.w3.org/1999/XMLSchema-
instance"
        Serializer.StartBody
        Serializer.StartElement strMethod, "urn:SystemsUnion-" & strComponent,
, "m"
        '****************************************
        '--- SECURITY VOUCHER ---
        Serializer.StartElement "oemVoucher"
        Serializer.SoapAttribute "xsi:type", "", "SOAPSDK1:string"
        Serializer.WriteString strVoucher
        Serializer.EndElement
        '****************************************
        Serializer.StartElement "strPayload"
        Serializer.WriteXml strPayload
        Serializer.EndElement
        Serializer.EndElement
        Serializer.EndBody
        'This line actually fires off the call to SSC.
        Serializer.EndEnvelope
    Connector.EndMessage

    'Check for TimeOut
    If Err.Number = 5400 Then
      InvokeSOAPCall = "Connection timed out"
      Connector.Reset
      Exit Function
    End If

    ' The following Reader object opens the SOAP Response
    ' and evaluates if any SOAP Fault was raised
    ' The SOAP Response is placed in the strResult variable
    Reader.Load Connector.OutputStream
    If Not Reader.Fault Is Nothing Then
        strResult = Reader.FaultString.Text
    Else
        strResult = Reader.RpcResult.Text
    End If

    'Reset the SOAP Connection
    Connector.Reset

    InvokeSOAPCall = strResult
```

```
End Function
```

Visual Basic .NET

Web References – no security manager

This goes in the main form code:

```
Private Sub btnGo_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGo.Click

    'Build input payload
    Dim sInputPayload As String
    sInputPayload = "<?xml version=""1.0"" encoding=""UTF-8"" ?>" & _
                    "<SSC>" & _
                    "  <User>" & _
                    "    <Name>PKP</Name>" & _
                    "  </User>" & _
                    "  <SunSystemsContext>" & _
                    "    <BusinessUnit>PKP</BusinessUnit>" & _
                    "  </SunSystemsContext>" & _
                    "  <Payload>" & _
                    "    <Address>" & _
                    "      <AddressCode>Demo1</AddressCode>" & _
                    "      <AddressLine1>My Street</AddressLine1>" & _
                    "      <TownCity>My Town</TownCity>" & _
                    "      <PostalCode>AB12 3CD</PostalCode>" & _
                    "      <TelephoneNumber>01234 567890</TelephoneNumber>" & _
                    "      <LookupCode>DEMO1</LookupCode>" & _
                    "    </Address>" & _
                    "  </Payload>" & _
                    "</SSC>"

     'Instantiate instance of Address Web Reference
     Dim Address As New AddressComponent.Address

    'Set it to point to desired SSC Server
    Address.Url = "http://" & TextBox1.Text & ":8080/ssc/servlet/SUGSoap"

    'Call SSC
    Try
        TextBox2.Text = Address.CreateOrAmend(sInputPayload)
    Catch ex As Exception
        TextBox2.Text = ex.Message.ToString()
    Finally
        Address = Nothing
    End Try

End Sub
```

Web References – with security manager

This goes in the main form code:

```
Private Sub btnGo_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGo.Click

    Dim sSSCVoucher As String
    'Create instance of Security Manager so we can get the Voucher for SSC
    Dim oSecMan As New SunSystems.Connect.Client.SecurityManager(TextBox1.Text)

    If oSecMan.Result.NOT_LOGGED_IN Then
        Try
            oSecMan.Login("PKP", "")
            If oSecMan.Authorised Then
                'All worked so save voucher!!
                sSSCVoucher = oSecMan.Voucher
            Else
                TextBox2.Text = "Password for user is incorrect."
```



```
            End If
        Catch ex As Exception
            TextBox2.Text = "An error occurred in Validation: " & oSecMan.ErrorMsg
        End Try
    End If

    'Destroy instance of Security Manager object
    oSecMan = Nothing

    'Build input payload
    Dim sInputPayload As String
    sInputPayload = "<?xml version=""1.0"" encoding=""UTF-8"" ?>" & _
                    "<SSC>" & _
                    "  <User>" & _
                    "    <Name>PKP</Name>" & _
                    "  </User>" & _
                    "  <SunSystemsContext>" & _
                    "    <BusinessUnit>PKP</BusinessUnit>" & _
                    "  </SunSystemsContext>" & _
                    "  <Payload>" & _
                    "    <Address>" & _
                    "      <AddressCode>Demo1</AddressCode>" & _
                    "      <AddressLine1>My Street</AddressLine1>" & _
                    "      <TownCity>My Town</TownCity>" & _
                    "      <PostalCode>AB12 3CD</PostalCode>" & _
                    "      <TelephoneNumber>01234 567890</TelephoneNumber>" & _
                    "      <LookupCode>DEMO1</LookupCode>" & _
                    "    </Address>" & _
                    "  </Payload>" & _
                    "</SSC>"

    'Instantiate instance of Address Web Reference
    Dim Address As New AddressComponent.Address

    'Set it to point to desired SSC Server
    Address.Url = "http://" & TextBox1.Text & ":8080/ssc/servlet/SUGSoap"

    'Call SSC
    Try
        TextBox2.Text = Address.CreateOrAmend(sSSCVoucher, sInputPayload)
    Catch ex As Exception
        TextBox2.Text = ex.Message.ToString()
    Finally
        Address = Nothing
    End Try

End Sub
```

C#

Web Reference – no security manager

This goes in the main form code:

```
private void btnGo_Click(object sender, System.EventArgs e)
{

       //Build input payload
       string sInputPayload;
       sInputPayload = "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" +
                       "<SSC>" +
                       "  <User>" +
                       "    <Name>PKP</Name>" +
                       "  </User>" +
                       "  <SunSystemsContext>" +
                       "    <BusinessUnit>PKP</BusinessUnit>" +
                       "  </SunSystemsContext>" +
                       "  <Payload>" +
                       "    <Address>" +
                       "      <AddressCode>Demo1</AddressCode>" +
                       "      <AddressLine1>My Street</AddressLine1>" +
                       "      <TownCity>My Town</TownCity>" +
                       "      <PostalCode>AB12 3CD</PostalCode>" +
                       "      <TelephoneNumber>01234
567890</TelephoneNumber>" +
                       "      <LookupCode>DEMO1</LookupCode>" +
                       "    </Address>" +
                       "  </Payload>" +
                       "</SSC>";

       //Instantiate instance of Address Web Reference
       AddressComponent.Address Address = new AddressComponent.Address();

       //Set it to point to desired SSC Server
       Address.Url = "http://" + textBox1.Text +
":8080/ssc/servlet/SUGSoap";

       //Call SSC
       try
       {
              textBox2.Text = Address.CreateOrAmend(sInputPayload);
       }
       catch (Exception ex)
       {
              textBox2.Text = ex.Message.ToString();
       }
       finally
       {
              //Destroy instance of Address Web Reference
              Address = null;
       }

}
```

Web Reference – with security manager

This goes in the main form code:

```
private void btnGo_Click(object sender, System.EventArgs e)
{

       string sSSCVoucher = "";
       //Create instance of Security Manager so we can get the Voucher for SSC
       SunSystems.Connect.Client.SecurityManager oSecMan = new

       SunSystems.Connect.Client.SecurityManager(textBox1.Text);
```



```
	if(oSecMan.Result ==
SunSystems.Connect.Client.AuthoriseResult.NOT_LOGGED_IN)
	{
		try
		{
			oSecMan.Login("PKP", "");
			if(oSecMan.Authorised)
			{
				//All worked so save voucher!!
				sSSCVoucher = oSecMan.Voucher;
			}
			else
			{
				textBox2.Text = "Password for user is incorrect.";
				return;
			}
		}
		catch (Exception ex)
		{
			textBox2.Text = "An error occurred in Validation: " +
					oSecMan.ErrorMsg;
			return;
		}
		finally
		{
			//Destroy instance of Security Manager
			oSecMan = null;
		}
	}

	//Destroy instance of Security Manager object
	oSecMan = null;

	//Build input payload
	string sInputPayload;
	sInputPayload = "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" +
			"<SSC>" +
			"  <User>" +
			"    <Name>PKP</Name>" +
			"  </User>" +
			"  <SunSystemsContext>" +
			"    <BusinessUnit>PKP</BusinessUnit>" +
			"  </SunSystemsContext>" +
			"  <Payload>" +
			"    <Address>" +
			"      <AddressCode>Demo1</AddressCode>" +
			"      <AddressLine1>My Street</AddressLine1>" +
			"      <TownCity>My Town</TownCity>" +
			"      <PostalCode>AB12 3CD</PostalCode>" +
			"      <TelephoneNumber>01234 567890</TelephoneNumber>" +
			"      <LookupCode>DEMO1</LookupCode>" +
			"    </Address>" +
			"  </Payload>" +
			"</SSC>";

	//Instantiate instance of Address Web Reference
	AddressComponent.Address Address = new AddressComponent.Address();

	//Set it to point to desired SSC Server
	Address.Url = "http://" + textBox1.Text + ":8080/ssc/servlet/SUGSoap";

	//Call SSC
	try
	{
		textBox2.Text = Address.CreateOrAmend(sSSCVoucher,sInputPayload);
	}
	catch (Exception ex)
	{
		textBox2.Text = ex.Message.ToString();
	}
	finally
	{
		//Destroy instance of Address Web Reference
		Address = null;
	}
}
```

Java

Before you can begin using the SSC Client Library from within Java, there are several things you must do.

When creating your Java project, ensure you include the following libraries:
<SunSystems5>\ssc\Lib\connect-client.jar
<SunSystems5>\ssc\Lib\xercesImpl.jar

Next, add the Runtime JRE – the correct version of which can be found in the <SunSystems5>\ssc\jre directory.

Finally, in many cases, Java requires a security policy to be specified on the command line.

No security manager

```
package demo;

import java.io.BufferedReader;
import java.io.InputStreamReader;

//Import SSC Client Library stuff from connect-client.jar
import com.systemsunion.ssc.client.*;

public class CallSSC {

       //Declare server and port variables
       static String HOST;
       static int PORT = 8080;

       public static void main(String[] args) {

              try {
                     BufferedReader in = new BufferedReader(
                                   new InputStreamReader(System.in));

                     // Get SSC Server
                     System.out.print("SSC Server: ");
                     System.out.flush();
                     HOST = in.readLine();

                     //Build input payload
                     String sInputPayload;
                     sInputPayload = "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" +
                        "<SSC>" +
                        "  <User>" +
                        "    <Name>PKP</Name>" +
                        "  </User>" +
                        "  <SunSystemsContext>" +
                        "    <BusinessUnit>PKP</BusinessUnit>" +
                        "  </SunSystemsContext>" +
                        "  <Payload>" +
                        "    <Address>" +
                        "      <AddressCode>Demo1</AddressCode>" +
                        "      <AddressLine1>My Street</AddressLine1>" +
                        "      <TownCity>My Town</TownCity>" +
                        "      <PostalCode>AB12 3CD</PostalCode>" +
                        "      <TelephoneNumber>01234 567890</TelephoneNumber>" +
                        "      <LookupCode>DEMO1</LookupCode>" +
                        "    </Address>" +
                        "  </Payload>" +
                        "</SSC>";

                     try {

                            //Instantiate instance of SoapComponent
                            SoapComponent ssc = new SoapComponent(HOST,PORT);

                            //Call SSC
                            String result = ssc.execute("Address", "CreateOrAmend",
sInputPayload);
```



```
                        System.out.println(result);

                    }
                    catch (Exception ex)  {
                        System.out.print("An error occurred logging in to
SunSystems:\r\n");
                        ex.printStackTrace();
                    }

            }
            catch (Exception ex) {
                ex.printStackTrace();
            }
        }
}
```

With security manager

```
package demo;

import java.io.BufferedReader;
import java.io.InputStreamReader;

//Import SSC Client Library stuff from connect-client.jar
import com.systemsunion.ssc.client.*;
//Import Security Manager
import com.systemsunion.ssc.client.security.SecurityProxy;


public class CallSSCWithSecMan {

        //Declare server and port variables
        static String HOST;
        static int PORT = 8080;

        public static void main(String[] args) {

            try {

                BufferedReader in = new BufferedReader(
                              new InputStreamReader(System.in));

                // Get SSC Server
                System.out.print("SSC Server: ");
                System.out.flush();
                HOST = in.readLine();

                //Create instance of Security Manager so
                //we can get the Voucher for SSC
                SecurityProxy secman = new SecurityProxy(HOST);
                String voucher = secman.login("PKP","").toString();

                //Build input payload
                String sInputPayload;
                sInputPayload = "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" +
                   "<SSC>" +
                   "  <User>" +
                   "    <Name>PKP</Name>" +
                   "  </User>" +
                   "  <SunSystemsContext>" +
                   "    <BusinessUnit>PKP</BusinessUnit>" +
                   "  </SunSystemsContext>" +
                   "  <Payload>" +
                   "    <Address>" +
                   "      <AddressCode>Demo1</AddressCode>" +
                   "      <AddressLine1>My Street</AddressLine1>" +
                   "      <TownCity>My Town</TownCity>" +
                   "      <PostalCode>AB12 3CD</PostalCode>" +
                   "      <TelephoneNumber>01234 567890</TelephoneNumber>" +
                   "      <LookupCode>DEMO1</LookupCode>" +
                   "    </Address>" +
                   "  </Payload>" +
                   "</SSC>";
```

```
                    try {

                           //Instantiate instance of SoapComponent
                           SoapComponent ssc = new SoapComponent(HOST,PORT);

                           //Apply security voucher
                           ssc.authenticate(voucher);

                           //Call SSC
                           String result = ssc.execute("Address", "CreateOrAmend",
sInputPayload);

                           System.out.println(result);

                    }
                    catch (Exception ex)  {
                           System.out.print("An error occurred logging in to
SunSystems:\r\n");
                           ex.printStackTrace();
                    }

             }
             catch (Exception ex) {
                    ex.printStackTrace();
             }
       }
}
```


Now that you have seen the code required to call SSC in various ways, it's time to try it for yourself…

EXERCISE 5 – Create an Address

In the language of your choice, create a program which accepts the following parameters:

- Address Code
- Address Line 1
- Address Line 2
- Address Line 3
- Town/City
- County
- Post Code
- Telephone Number

Send this address record into SunSystems via SSC.

Verify that the record exists in SunSystems and that it is accurate.

Be as creative as you like with the look and feel of the application.

EXERCISE 6 – Querying an Address

Amend your application to add a "search" function, whereby you can enter an Address Code and it performs an SSC Query for that Address Code. Show the results of the query on-screen. If you wish you can display the SSML returned, or if you feel more adventurous, parse the XML and present the data returned in a more user friendly fashion.

EXERCISE 7 – Deleting an Address

Amend your application again, and add a "delete" option, whereby you can enter an Address Code and use SSC to delete it.

> **Note:** Whether or not you are allowed to delete an Address Record depends on SunSystems business logic, so you may find some addresses cannot be deleted. But if you have just created one, and not yet linked it to an account or any other part of SunSystems, you should be able to delete it.

Chapter 6 – Basic Error Handling

Returned Payloads

Example Returned Payloads

Below is an example of returned XML from the payloads generated in the programs above:


```
<?xml version="1.0" encoding="UTF-8"?>
<SSC>
    <User>
        <Name>PKP</Name>
    </User>
    <SunSystemsContext>
        <BusinessUnit>PKP</BusinessUnit>
    </SunSystemsContext>
    <Payload>
        <Address status="success">
            <AddressCode>Demo1</AddressCode>
        </Address>
    </Payload>
</SSC>
```


You will find most of the SSC Components return information in this way. The above example is that of a success. In the case of failure the XML returned would resemble the following


```
<?xml version="1.0" encoding="UTF-8"?>
<SSC>
    <User>
        <Name>PKP</Name>
    </User>
    <SunSystemsContext>
        <BusinessUnit>PKP</BusinessUnit>
    </SunSystemsContext>
    <Payload>
        <Address status="fail" Reference="2" rejected="true">
            <AddressCode>Demo1</AddressCode>
            <AddressLine1>Line1</AddressLine1>
            <CountryCode Reference="1">FRANCE</CountryCode>
            <LookupCode>DEMO1</LookupCode>
            <PostalCode>AB12 3CD</PostalCode>
            <TelephoneNumber>01234 567890</TelephoneNumber>
            <TownCity>My Town</TownCity>
            <Messages>
                <Message Level="error" Reference="1">
                    <UserText>The value 'FRANCE' is not valid and could not be entered.
The external string value 'FRANCE' is too long for the internal field (internal field length is
3).</UserText>
                    <Application>
                        <Component>Address</Component>
                        <DataItem>Address.CountryCode</DataItem>
                        <Driver>SASI</Driver>
                        <LastMethod>Enter(&quot;Address.CountryCode&quot;, 0, 110,
&quot;SKIP_IF_NO_DATA&quot;)</LastMethod>
                        <Method>CreateOrAmend</Method>
                        <Value>FRANCE</Value>
                        <Version>63</Version>
                    </Application>
                </Message>
                <Message Level="error" Reference="2">
```

```
                <UserText>The import of this record was aborted due to errors.    The value
'FRANCE' is not valid and could not be entered.    The value 'FRANCE' is not valid and could
not be entered.    The external string value 'FRANCE' is too long for the internal field (internal
field length is 3).</UserText>
                <Application>
                    <Component>Address</Component>
                    <DataItem>Address.CountryCode</DataItem>
                    <Driver>SASI</Driver>
                    <LastMethod>Enter(&quot;Address.CountryCode&quot;, 0, 110,
&quot;SKIP_IF_NO_DATA&quot;)</LastMethod>
                    <Method>CreateOrAmend</Method>
                    <Value>FRANCE</Value>
                    <Version>63</Version>
                </Application>
            </Message>
        </Messages>
    </Address>
  </Payload>
</SSC>
```


Parsing

Once you receive the returned payload, you will need to parse it to look for key indicators as to whether it has succeeded or failed.

There are two rules which dictate how this information is identified. As you can see in the examples above, the first key thing to look for is for each record, there is a “status” attribute. This will be set to either “success” or “fail”.

In the case of a failure, there will also be a “reference” attribute with a numeric value which relates to an error message contained within the line data. Additionally, depending on the type of error, you will see a reference also on the offending field.

Once an error has been determined, you should look for a child node called <Messages> - and within this – individual <Message> nodes for each error relating to the line. The Reference attribute of the <Message> node can be linked back to the “reference” attribute previously read in. This way you can then look for the <UserText> tag and report it back to the user, or log it – depending on your needs.

There is also an <Application> node within the <Message> node, which contains information about the Component where the error was generated. It can tell you the Component being run, the Method, driver type, data item, and component version. Again, this can be extracted and fed back to the user, logged, or ignored, depending on your desired program logic.

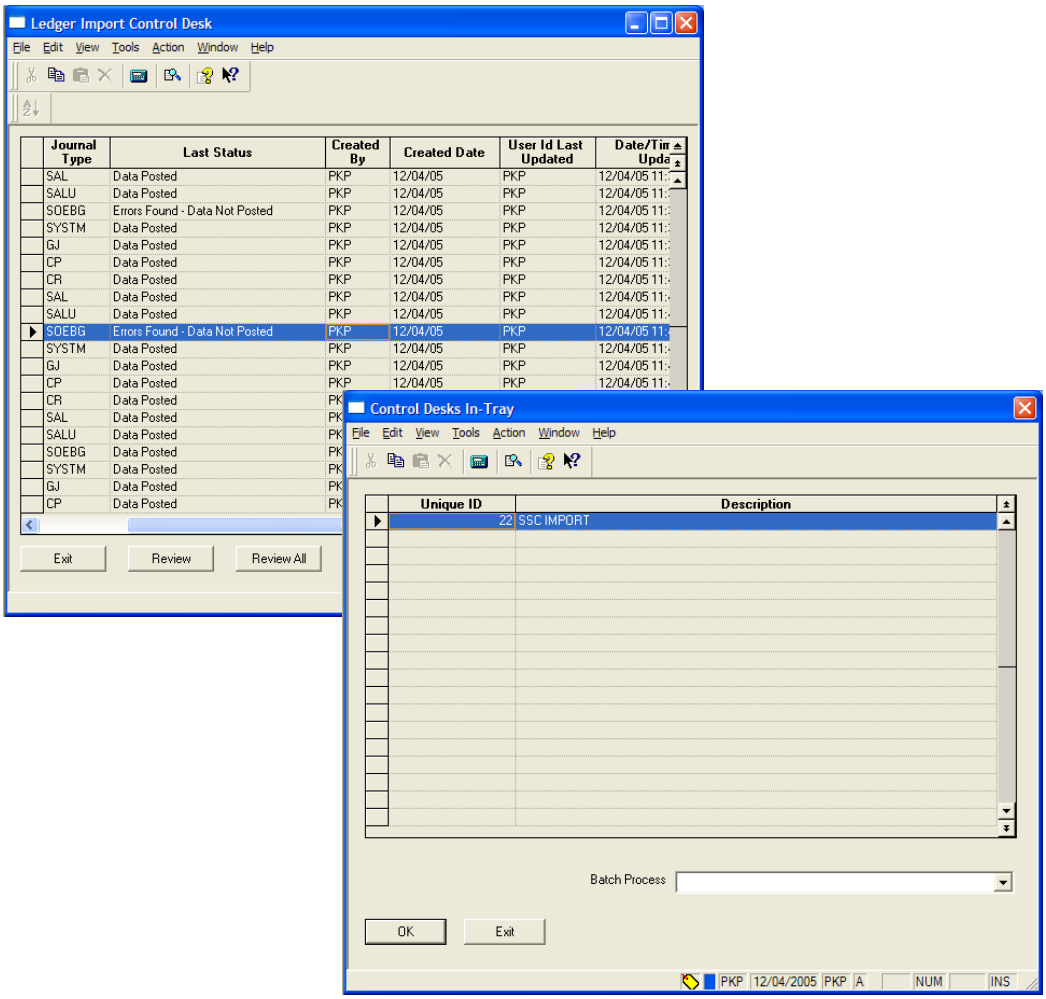
Ledger-specific error handling

When you are importing Ledger information into SunSystems the payloads returned will contain an extra node within the <Payload> tags, called <PostingHeaderDetails>. They look like this:


```
  <PostingHeaderDetails>
    <PostingHeaderId>22</PostingHeaderId>
    <PostngHdr_LastStatus>212</PostngHdr_LastStatus>
  </PostingHeaderDetails>
```


This gives you additional information about the import and should be referred to when importing Ledger data into SunSystems via. SSC, instead of the <Messages> nodes.

The <PostingHeaderId> tag contains the Ledger Import posting header ID, assigned by the SunSystems Ledger Import function, it can be used to link back to the import from the Ledger Import control desk within SunSystems:



The <PostingHdr_LastStatus> tag may have any of the following values (those in bold being the only ones normally expected when a transaction has been processed):

0	Data Not Yet Processed
1	**Validate Only - No Errors Found**
2	**Validate Only - Errors Found**
111	**Data Posted**
112	**Errors Found - Data still Posted**
121	**Data Posted to Hold**
122	**Errors Found - Data still Posted to Hold**
211	**No Errors - Data Posted**
212	**Errors Found - Data Not Posted**
221	**No Errors - Data Posted to Hold**
222	**Errors Found - Data Not Posted to Hold**
999	Currently Being Imported

Note: This information only refers to the last journal imported. If you had multiple journal types in your input payload resulting in multiple journals, this only refers to the last one posted. It does not refer to the entire import.

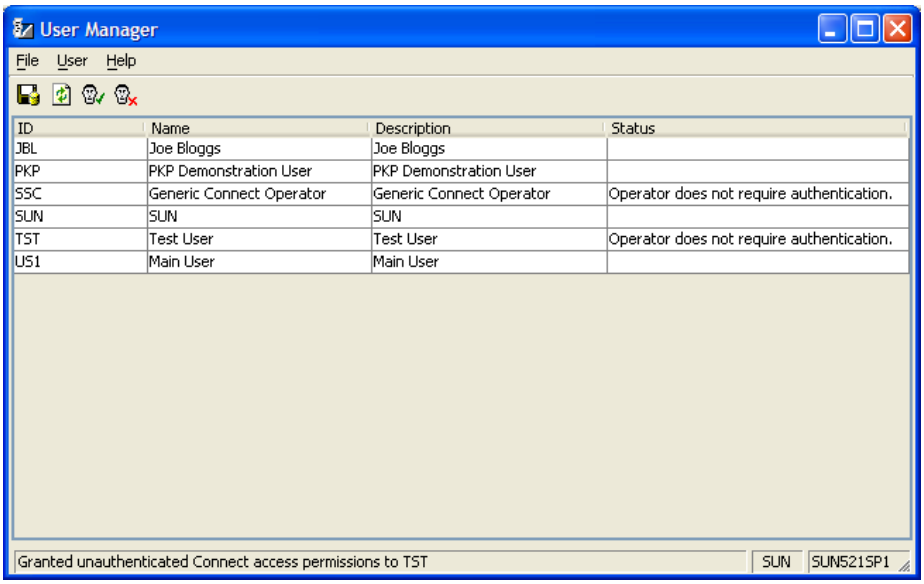
Chapter 7 – SunSystems Connect Tools & Administration Utilities

User Manager

Accessed via the 'UM' option in Navigation Manager, the User Manager application can be used to disable authentication for specific SunSystems users. The ID of a user who has been granted access through User Manager becomes a free pass to all Connect access.

This system was introduced to remove the need to include SunSystems passwords in the payload, which if intercepted could be used to gain access to interactive SunSystems applications. The effect of this was that the Connect interfaces became less secure, requiring local access restriction (and ideally locking down user permissions using Data Access controls).

The security model introduced in SunSystems 5.2 removes the need to grant access with user manager, but it is retained to support applications that have not yet been updated to use the new interfaces. Users should be encouraged to remove User Manager access to operators as soon as possible, and application developers should encourage this by implementing the new login procedure.



User Manager works by reading the operators directly from SunSystems (as configured in the OS=Operator Setup function) and displays them in a list. You simply select the operator you want to be able to use SSC in an unauthenticated fashion and click the Grant Access button:

To remove permissions from an operator, select the required operator from the list and click the Revoke Access button:

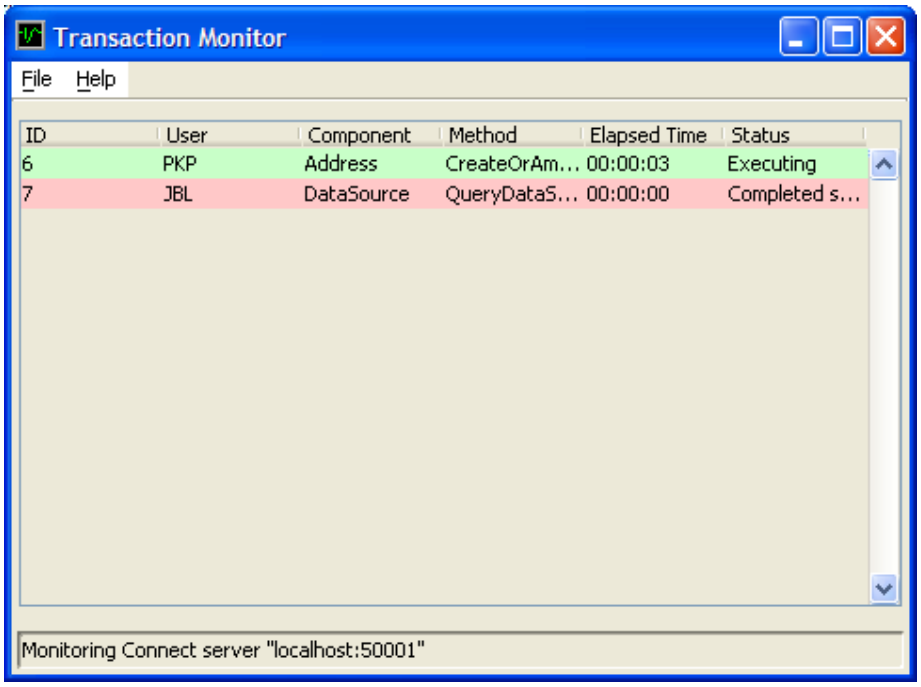
The 'Status' column on-screen shows you if the user is set to require authentication or not.

> **Note:** You need to be logged on to SunSystems as a Super User to be able to run SSC User Manager.

You are no longer required to re-start the SunSystems Connect Service for changes here to take effect. SSC picks these options up dynamically.

Transaction Monitor

Transaction Monitor displays a real-time view of the SSC activity on the SunSystems application server.



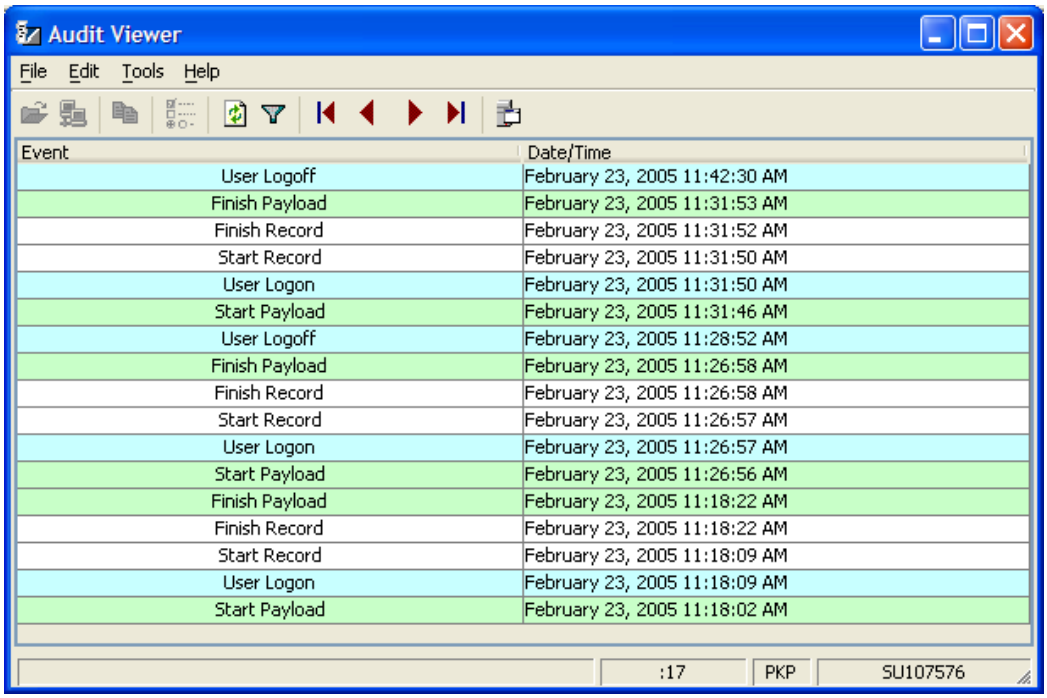
This can be extremely useful from a troubleshooting point of view as it allows you to see what processes are running currently, and their status.

Audit Viewer

Audit Viewer keeps a log of all transactions made via. SSC. From initialisation of calls, through to the user logging on, processing transactions, exceptions, and user logging off.

> **Note:** Audit Viewer only displays your 'own' transactions, unless you are logged on as the SUN SuperUser.

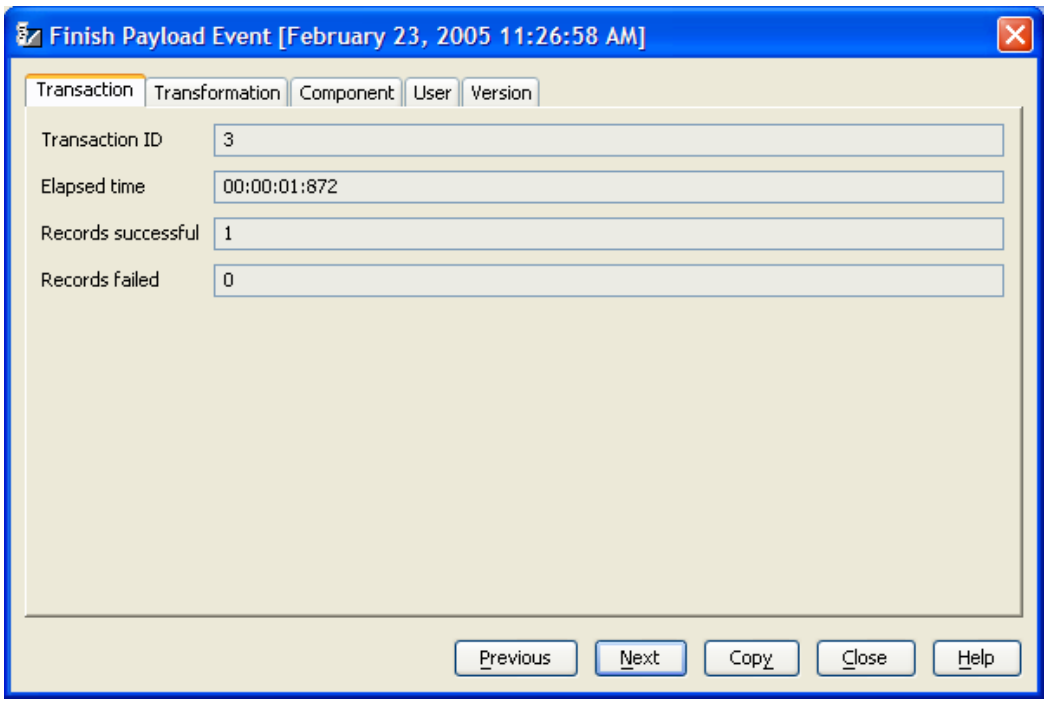
This is the main Audit Viewer screen:



Event	Date/Time
User Logoff	February 23, 2005 11:42:30 AM
Finish Payload	February 23, 2005 11:31:53 AM
Finish Record	February 23, 2005 11:31:52 AM
Start Record	February 23, 2005 11:31:50 AM
User Logon	February 23, 2005 11:31:50 AM
Start Payload	February 23, 2005 11:31:46 AM
User Logoff	February 23, 2005 11:28:52 AM
Finish Payload	February 23, 2005 11:26:58 AM
Finish Record	February 23, 2005 11:26:58 AM
Start Record	February 23, 2005 11:26:57 AM
User Logon	February 23, 2005 11:26:57 AM
Start Payload	February 23, 2005 11:26:56 AM
Finish Payload	February 23, 2005 11:18:22 AM
Finish Record	February 23, 2005 11:18:22 AM
Start Record	February 23, 2005 11:18:09 AM
User Logon	February 23, 2005 11:18:09 AM
Start Payload	February 23, 2005 11:18:02 AM

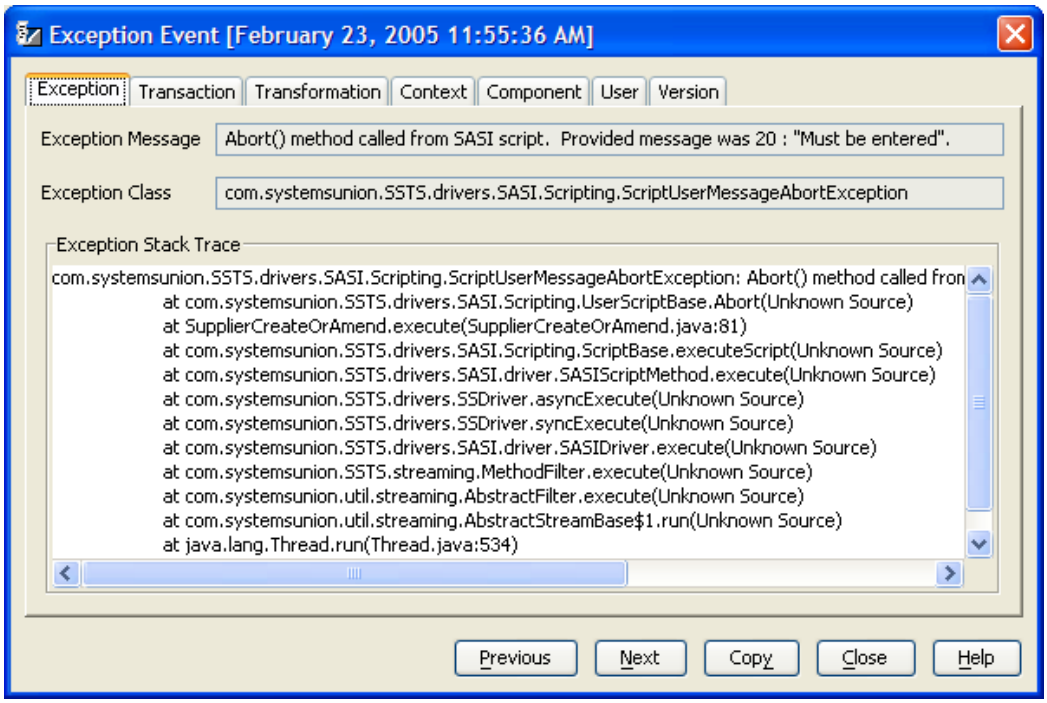
Each type of event is colour coded for ease-of-reading.

You can filter for various types of events, and double-clicking brings up details for the selected line:

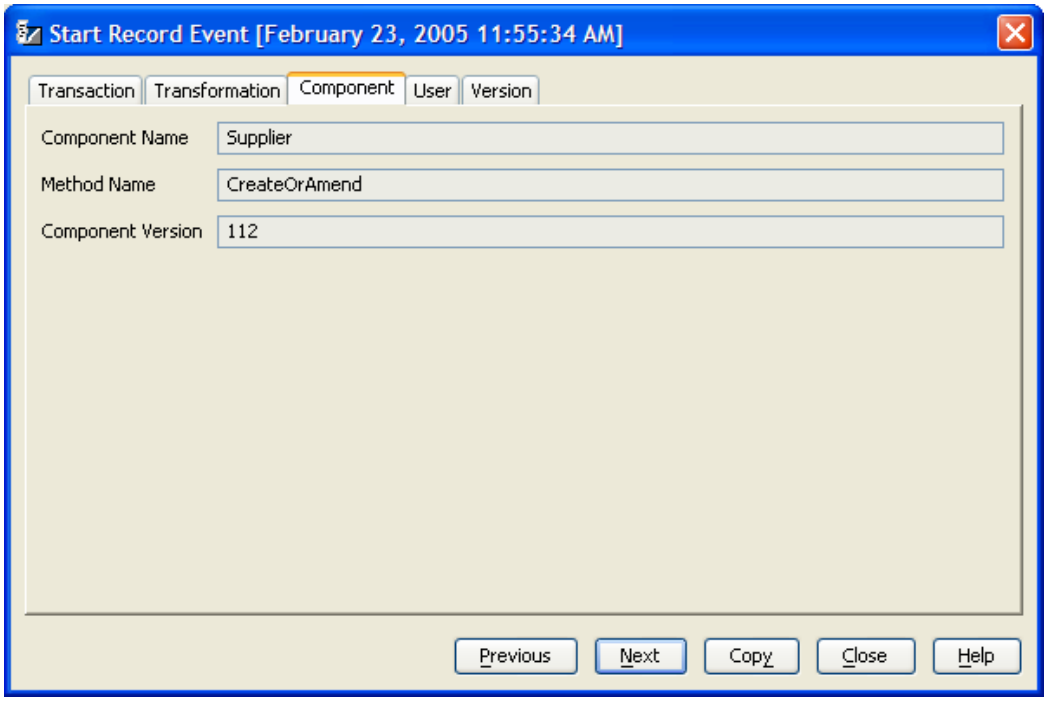


Each tab contains different information regarding the log entry, and depending on the type of entry, different tabs may be displayed accordingly.

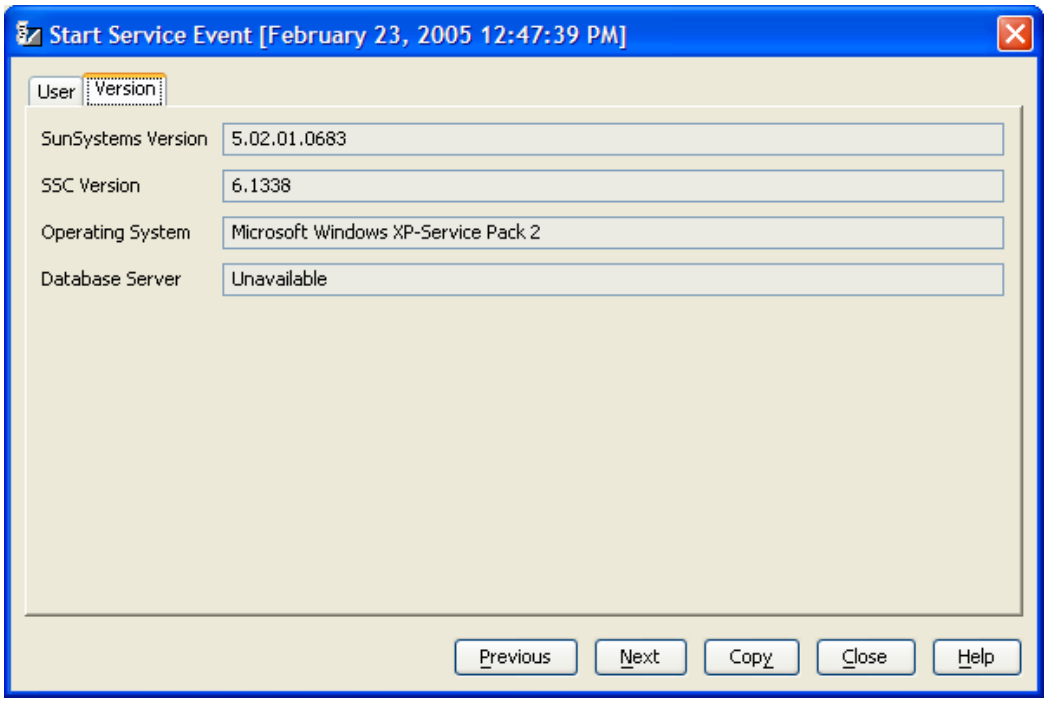
For example, when an exception has occurred, you can view the stack trace:



You can view Component information:



And you can view basic operating system parameters:



All this information can be copied at the touch of a button, by clicking the “Copy” button for a given event. This enables all of the details to quickly be copied into for example an e-mail to SunSystems support, thus:

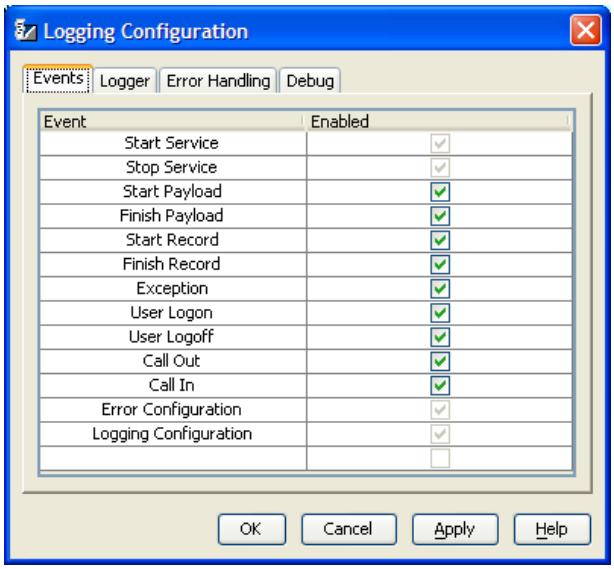
```
SSC Exception Event at 1118335093191
====================================

Exception Message     : Abort() method called from SASI script.  Provided message was
2788 : ''Item Code "11003" does not exist
PurchaseOrderReference PO/000077''.

Exception          Class                                                          :
com.systemsunion.SSTS.drivers.dsi.scripting.ScriptUserMessageAbortException

Exception Stack Trace :

com.systemsunion.SSTS.drivers.dsi.scripting.ScriptUserMessageAbortException:   Abort()
method called from SASI script.  Provided message was 2788 : ''Item Code "11003" does
not exist
PurchaseOrderReference PO/000077''.
        at         com.systemsunion.SSTS.drivers.dsi.scripting.ScriptBase.Abort(Unknown
Source)
        at PurchaseOrderCreate__DSI.ErrorCheck(PurchaseOrderCreate__DSI.java:1531)
        at
PurchaseOrderCreate__DSI.EnterPurchaseOrderLine(PurchaseOrderCreate__DSI.java:558)
        at PurchaseOrderCreate__DSI.execute(PurchaseOrderCreate__DSI.java:352)
        at
com.systemsunion.SSTS.drivers.dsi.scripting.ScriptBase.executeScript(Unknown Source)
        at com.systemsunion.SSTS.drivers.dsi.driver.DSIMethod.execute(Unknown Source)
        at com.systemsunion.SSTS.drivers.SSDriver.asyncExecute(Unknown Source)
        at com.systemsunion.SSTS.drivers.SSDriver.syncExecute(Unknown Source)
        at com.systemsunion.SSTS.drivers.dsi.driver.DSIDriver.execute(Unknown Source)
        at com.systemsunion.SSTS.streaming.MethodFilter.execute(Unknown Source)
        at com.systemsunion.util.streaming.AbstractFilter.execute(Unknown Source)
        at com.systemsunion.util.streaming.AbstractStreamBase$1.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:534)


Exception Context    :

Value                : 0
MessageNumber        : 2788
Version              : 167
Message              : Item Code "11003" does not exist
PurchaseOrderReference PO/000077
Method               : CreateOrAmend
Component            : PurchaseOrder
DataItem                                                                            :
PurchaseOrder.PurchaseOrderLine.VLAB1.Base.VPolVlabEntry_ConvRate
Driver               : DSI
LastMethod           : GetFormValue(0, 34)
Transaction ID       : 12
Elapsed Time         : 00:00:04:310
Records Successful   : 0
Records Failed       : 0
Component Name       : PurchaseOrder
Method Name          : CreateOrAmend
Component Version    : 167
Transfer Profile     : Unavailable
Source Datatype      : Unavailable
Target Datatype      : Unavailable
Transformation       : Unavailable
Customer Name        : Dev Serialisation 3 MonthFuse
SunSystems Serial No : 123456
User Name            : PKP
SunSystems Version   : 5.02.01.0683
SSC Version          : 6.1338
Operating System     : Microsoft Windows XP-Service Pack 2
DBMS Version         : Unavailable
```


Audit Viewer allows you to set up configuration settings to your requirements. If you are logged on to SunSystems as the SUN Super User, you can access the Edit > Settings menu option. This displays to you a configuration dialog box in which you can set various options regarding error logging.

Four tabs are displayed if your system is run on the application server; only Debug is available if you are logged onto a client.

Events

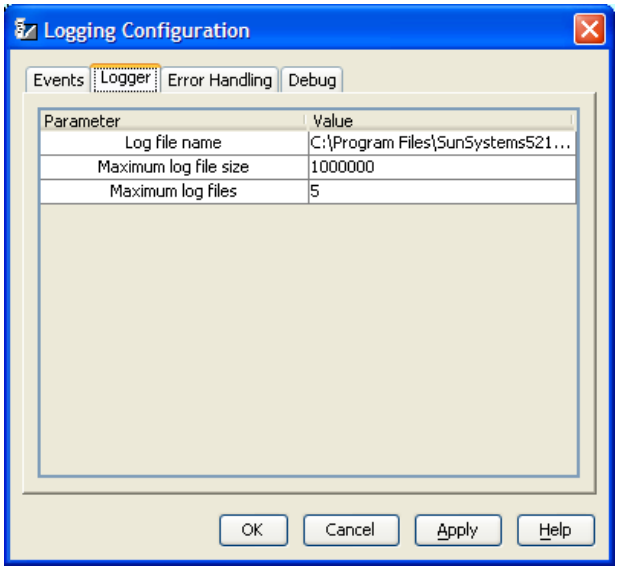
This tab is only displayed if your system is run on the application server.



Check the check boxes of the required events to be displayed. Start Service, Stop Service, Error Configuration and Logging Configuration are selected by default and cannot be disabled.

Logger

This tab is only displayed if your system is run on the application server.



You can update the default parameter values if required. Double-click on the required value to amend it.

Log file name

By default the Log file name is audit.dat and is located in the ssc/logging folder of your SunSystems root directory.

Maximum log file size

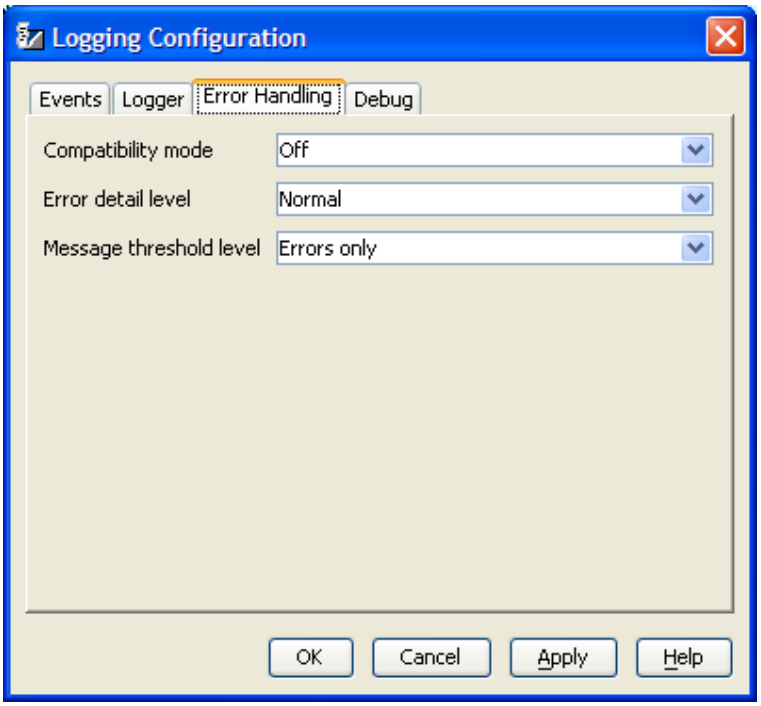
By default the maximum log file size is set to 1000000 (bytes). There is no limit to the size you can enter in this field.

Maximum log files

By default the maximum number of log files is set to 5. There is no limit to the number you can enter in this field.

Error Handling

This tab is only displayed if your system is run on the application server.



This tab enables you to select the default level of information provided in error messages.

Note: These settings can be overridden on a per-call basis by including the <ErrorContext> section of XML code in input payloads being submitted to SSC. This is further documented in Chapter 8 – Debugging SSC.

Compatibility mode

The default value of the compatibility mode is set to On, this means that any error messages generated are placed with the line that generated the errors and at the end of the payload.

Error detail level

The default value for error message detail is Normal. You can amend the setting to one of the following, as required:

- Limited - error messages only are displayed
- Normal - limited and context information
- Extended - normal and any exception stack trace
- Debug - not currently used.

Any changes made, apply globally and with immediate effect.

Be aware that if you choose to amend Error Message Detail Level to Extended or Debug, performance may be adversely affected.

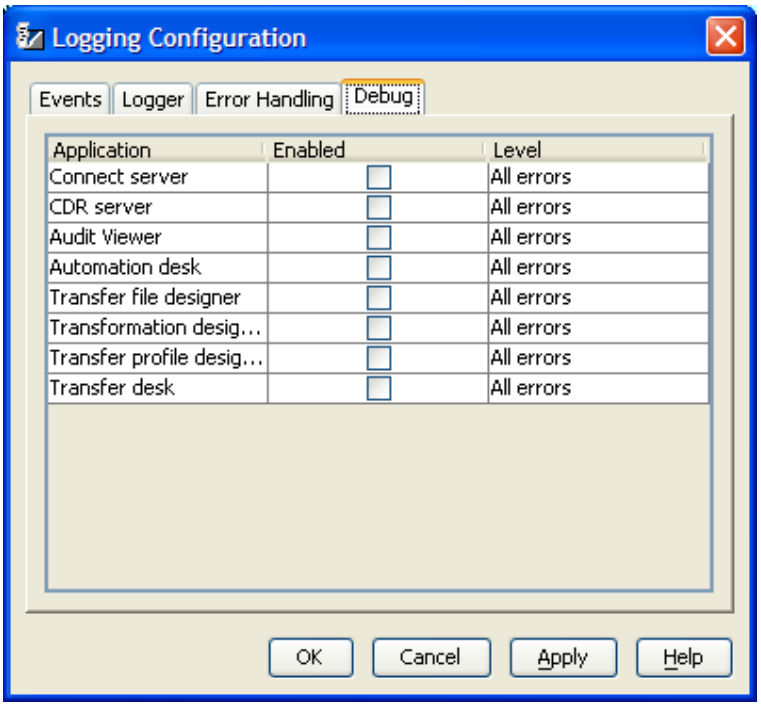
Message Threshold Level

This controls at what point messages are passed back in the output payloads from SSC. The default is for errors only to be reported but you can specify the following, as required:

- Errors only
- Warnings & errors

Debug

This tab allows you to turn on logging for the required parts of SSC & Transfer Desk.



Check the Enabled check box against the required Transfer Desk functions:

- Connect Server - only available if run on the application server
- CDR Server - only available if run on the application server
- Audit Viewer

- Automation Desk
- Transfer File Designer
- Transformation Designer
- Transfer Profile Designer
- Transfer Desk.

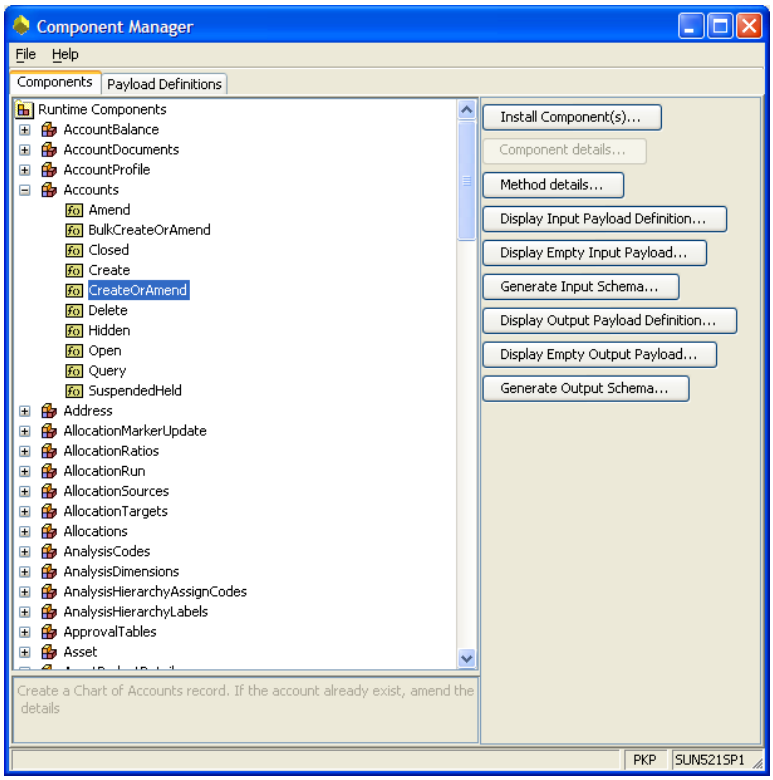
Select the level of debug required from the drop down list. Click OK. The log files are generated and saved in the <SunSystems5>\ssc\logging directory on the application server by default.

Logging is further documented in Chapter 8 – Debugging SSC.

Component Manager

The Component Manager has two main views:-

Components



This shows all the components and methods that are available at run-time.

One of the main uses for Component Manager is to install new components on to the SSC Server. Clicking the "Install Component(s)…" button shows a dialog box asking for the .jar file to install. A version check is done and if everything is okay, the new component is installed on top of the existing one.

Components are usually only deployed with installations of SunSystems, Service Packs, or occasionally, in patches (which must be requested from Systems Union). You cannot write or install your own 'components'.

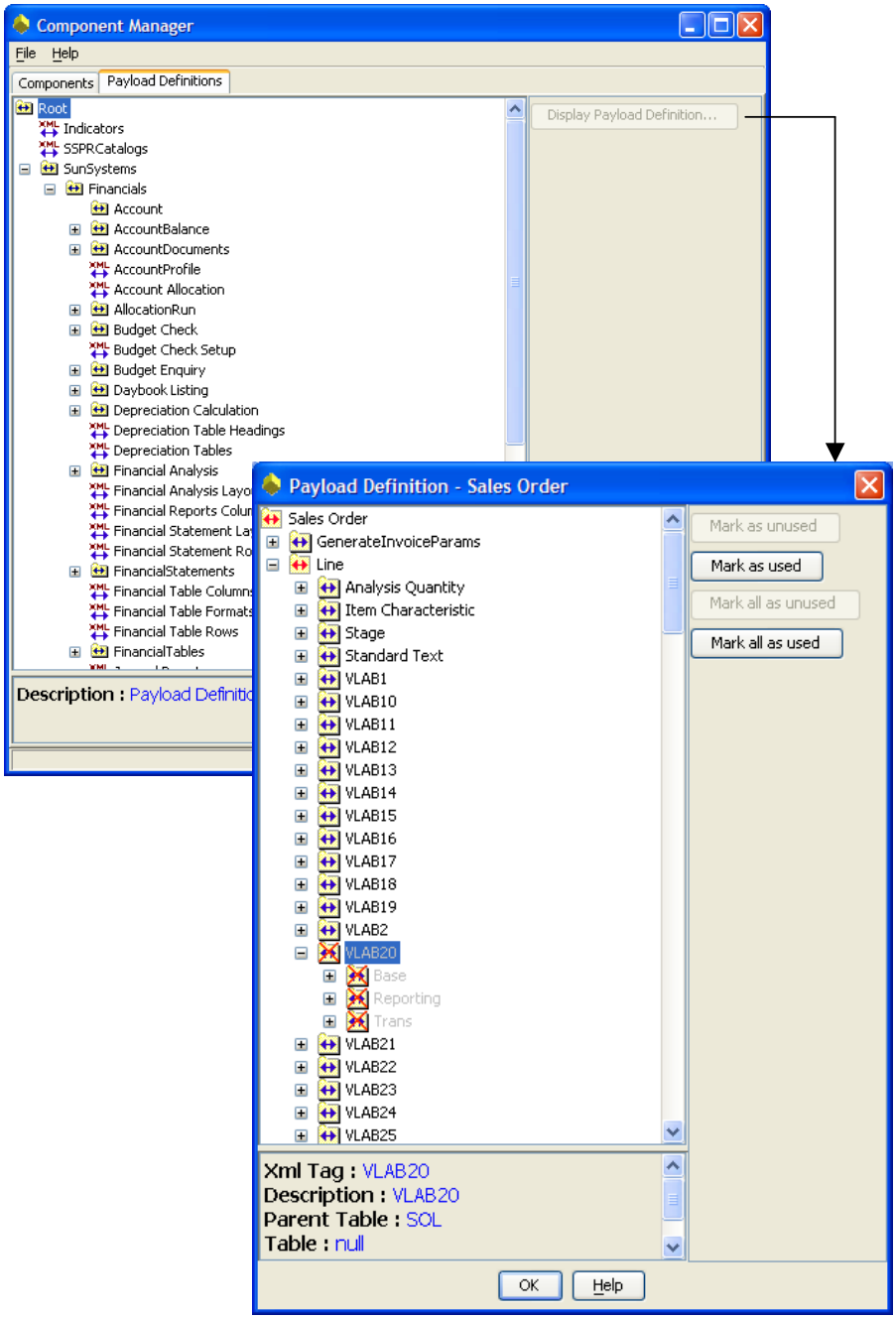
When accessing the methods there are a number of facilities you can use:-

Method Details
This gives a brief description of what the method will do.

Display Input and Output Payload Definitions
Shows the database hierarchy including fieldnames and their corresponding XML tag names.

Generate Input and Output Schemas
Allows the user to save input and output schemas to an XML file. The schema is rather large but does hold some important information like field sizes, valid values, etc.

Payload Definitions

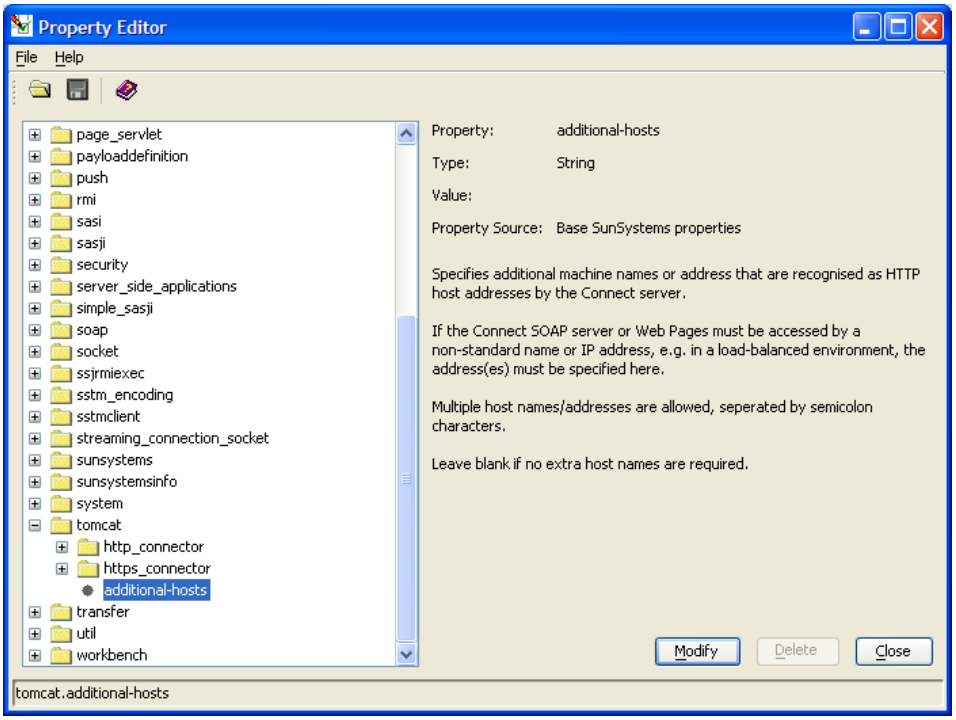


This functionality is for the tailoring of payloads. If you have a client who is not using all the functionality within SunSystems – for example Analysis – then this could be switched off.

This will improve performance because not as many tables will need to be referenced to find or update data. This has to be done with great care though as this will affect all SunSystems Connect (SSC), Transfer Desk and Common Data Retrieval (CDR) functions.

A prime example of where this tailoring is required is Value Labels in Order Fulfilment. There are 50 value labels, but most customers will only use a handful of these so these can be cut down and improve performance significantly.

Property Editor



Property Editor is a powerful tool which provides access to core SSC settings.

> **Note:** Always make a backup of user_props.xml before you modify any settings. Editing property values can have serious implications for your system and could cause problems that may require you to reinstall SunSystems. We cannot guarantee that problems resulting from the incorrect use of Property Editor can be solved. Use Property Editor at your own risk.

There are however a few settings in here which are very useful to know about.

DJI

num_concurrent_transactions
: Number of transactions that can run concurrently throught the DJI driver. When this number is exceeded, any new requests will be queued. The default value is 3. Larger numbers may give greater throughput but will require more memory and processor power.

Export

limit_row_count
: Flag that requests that the row count is set to a limiting number. This affects all export transactions, so should not usually be set to 'true'. If large output payloads are causing problems, this property, along with the 'max_result_set_row_count' property can be overriden by adding an OutputLimit tag to the Export input payload.

indent_xml
: Flag that controls export output xml formatting When formatting is turned on (=true), the additional characters make the payload significantly larger, and this can have a major impact on performance. You should leave this set to

false unless you need to examine output payloads in a text editor (e.g. Notepad). You can view the unformatted text in XML aware applications, e.g. Internet Explorer or XMLSpy.

max_result_set_row_count
The maximum number of rows that can be retrieved from the database during an Export. If limit_row_count = 'false', this value is ignored and an unlimited number of rows may be exported.

num_concurrent_transactions
Number of export transactions that can run concurrently. When this number is exceeded, any new export requests will be queued. The default is 3. Larger numbers may give greater throughput but will require more memory and processor power.

SASI

logoff_delay_secs
This is the number of seconds that a SASI session remains logged in for, after its processing is complete. This can be used to extend or troubleshoot SASI timeout issues.

num_concurrent_transactions
Number of transactions that can run concurrently throught the SASI driver. When this number is exceeded, any new requests will be queued. The default value is 3. Larger numbers may give greater throughput but will require more memory and processor power. It is recommended that this number does not exceed 5 on medium specification servers.

logons_per_opr
Number of logons per operator object. SunSystems limits an operator to 9 sessions, and we need one to logoff with, so good values are 1..8, but 8 will give best performance

Security

secure_sockets
Controls communication between the Connect Server and client applications. If true, RMI and other communications are secured with Secure Sockets (SSL). If false, simple unencrypted sockets are used. If the connections occur behind a secure firewall, this may be acceptable and will improve performance.

SOAP

enabled
If set to false, disables the SOAP service.

server_URL
URL of the address to post SOAP messages to.

urn_prefix
Prefix used for the name of the deployment service on the soap server.

Tomcat

additional-hosts
Specifies additional machine names or address that are recognised as HTTP host addresses by the Connect server. If the Connect SOAP server or Web Pages must be accessed by a non-standard name or IP address, e.g. in a load-balanced environment, the address(es) must be specified here. Multiple

host names/addresses are allowed, separated by semicolon characters. Leave blank if no extra host names are required.

Transfer

logging_directory

Specifies where Transfer Desk log files are output to.

Online Help Text

There are two help files included with SunSystems 5 which can help you in creating payloads and calling SSC.

- 01SunSystemsConnect.chm
- 01SSCTechRef.chm

The first one, 01SunSystemsConnect.chm, is a general document covering the administration tools described above.

01SSCTechRef.chm is a more detailed programmer's guide to the SSC architecture, XML, and component documentation.

Chapter 8 – Debugging SSC

Errors in returned payload

Many errors which occur within SSC are incorporated into the output payload sent back to the calling application by SSC. As previously discussed, this is how a typical error might look when returned from SSC:


```
<?xml version="1.0" encoding="UTF-8"?>
<SSC>
    <User>
        <Name>PKP</Name>
    </User>
    <SunSystemsContext>
        <BusinessUnit>PKP</BusinessUnit>
    </SunSystemsContext>
    <Payload>
        <Address status="fail" Reference="2" rejected="true">
            <AddressCode>Demo1</AddressCode>
            <AddressLine1>Line1</AddressLine1>
            <CountryCode Reference="1">FRANCE</CountryCode>
            <LookupCode>DEMO1</LookupCode>
            <PostalCode>AB12 3CD</PostalCode>
            <TelephoneNumber>01234 567890</TelephoneNumber>
            <TownCity>My Town</TownCity>
            <Messages>
                <Message Level="error" Reference="1">
                    <UserText>The value 'FRANCE' is not valid and could not be entered.
The external string value 'FRANCE' is too long for the internal field (internal field length is
3).</UserText>
                    <Application>
                        <Component>Address</Component>
                        <DataItem>Address.CountryCode</DataItem>
                        <Driver>SASI</Driver>
                        <LastMethod>Enter(&quot;Address.CountryCode&quot;, 0, 110,
&quot;SKIP_IF_NO_DATA&quot;)</LastMethod>
                        <Method>CreateOrAmend</Method>
                        <Value>FRANCE</Value>
                        <Version>63</Version>
                    </Application>
                </Message>
                <Message Level="error" Reference="2">
                    <UserText>The import of this record was aborted due to errors.  The value
'FRANCE' is not valid and could not be entered.  The value 'FRANCE' is not valid and could
not be entered.  The external string value 'FRANCE' is too long for the internal field (internal
field length is 3).</UserText>
                    <Application>
                        <Component>Address</Component>
                        <DataItem>Address.CountryCode</DataItem>
                        <Driver>SASI</Driver>
                        <LastMethod>Enter(&quot;Address.CountryCode&quot;, 0, 110,
&quot;SKIP_IF_NO_DATA&quot;)</LastMethod>
                        <Method>CreateOrAmend</Method>
                        <Value>FRANCE</Value>
                        <Version>63</Version>
                    </Application>
                </Message>
            </Messages>
        </Address>
    </Payload>
</SSC>
```


Let us dissect this payload:


```
<Address status="fail" Reference="2" rejected="true">
```


This indicates that there was a failure importing this address record. It gives us a reference number of 2. So we can look up message number 2 within the <Messages> node within this Address record:


```
<Message Level="error" Reference="2">
<UserText>The import of this record was aborted due to errors. The value 'FRANCE' is not valid and could not be entered. The value 'FRANCE' is not valid and could not be entered. The external string value 'FRANCE' is too long for the internal field (internal field length is 3).</UserText>
<Application>
<Component>Address</Component>
<DataItem>Address.CountryCode</DataItem>
<Driver>SASI</Driver>
<LastMethod>Enter(&quot;Address.CountryCode&quot;, 0, 110, &quot;SKIP_IF_NO_DATA&quot;)</LastMethod>
<Method>CreateOrAmend</Method>
<Value>FRANCE</Value>
<Version>63</Version>
</Application>
</Message>
```


It is clear that the error being generated is from the value “FRANCE” in the CountryCode field.


```
<CountryCode Reference=”1”>FRANCE</CountryCode>
```


This indicates that this field failed to import the country code “FRANCE”. Again it gives us a reference number, this time it is 1.


```
<Message Level="error" Reference="1">
<UserText>The value 'FRANCE' is not valid and could not be entered. The external string value 'FRANCE' is too long for the internal field (internal field length is 3).</UserText>
<Application>
<Component>Address</Component>
<DataItem>Address.CountryCode</DataItem>
<Driver>SASI</Driver>
<LastMethod>Enter(&quot;Address.CountryCode&quot;, 0, 110, &quot;SKIP_IF_NO_DATA&quot;)</LastMethod>
<Method>CreateOrAmend</Method>
<Value>FRANCE</Value>
<Version>63</Version>
</Application>
</Message>
```


In this case, it is essentially the same message being output as before. But reference 1 is the individual field error – whereas reference 2 is the overall record error message.

Note that in both cases, there is an <Application> node, detailing the component that failed and some more low-level information such as component version, the data item, and if pertinent, SASI debug information.

Using the <ErrorContext> element in SSML

This high-level element contains the elements that control various aspects of error messaging and logging.

The following excerpt shows the structure of the <ErrorContext. element.

```
<SSC>
    <ErrorContext>
        <CompatibilityMode></CompatibilityMode>
        <ErrorOutput></ErrorOutput>
        <ErrorThreshold></ErrorThreshold>
    </ErrorContext>
</SSC>
```

This can be included in your input SSML payload.

CompatibilityMode

Overrides the SSC default for message format:

- Set to 0 to force all messages to be generated in the SunSystems v5.2.n format.
- Set to 1 to force all messages to be generated in the SunSystems v5.1.n format.

Example: `<CompatibilityMode>1</CompatibilityMode>`

ErrorOutput

Overrides the SSC default for the amount of message detail that is generated by SSC whenever a message is to be sent:

- Set to 0 to generate messages with a limited level of detail.
- Set to 1 to generate messages with a normal level of detail
- Set to 2 to generate messages with a extended level of detail
- Set to 3 to generate messages with a debug level of detail.

Example: `<ErrorOutput>2</ErrorOutput>`

ErrorThreshold

Overrides the SSC default for the types of messages that appear in the output payload:

- Set to 0 to include warning and error messages
- Set to 1 to include error messages only

Example: `<ErrorOutput>1</ErrorOutput>`

Errors returned in the XML payload will reflect the settings to the above elements.

SASI Scripts & SASI Views

Scripts

As already briefly outlined in chapter 2, SASI works by following a script, and running SunSystems in a background mode, to programmatically 'drive' the forms and interface.

The scripts are stored within the component .jar files. Here is an example of part of the script for creating Addresses:

```
Proc main()
{
	/*
		SASI script to create/amend addresses
	*/

	// Open function using the specially created form
	OpenFunction("VW5138", "SASI");

	// Loop through all Address entries in the input payload
	while (DataIsAvailable("Address"))
	{
		Enter("Address.AddressCode", 0, ADDRESS_CODE, ERROR_IF_NO_DATA);
		if (CurrentMessage() != -1)
		{
			Abort();
		}
		Press("OK");

		if (CurrentMessage() == DOES_NOT_EXIST)
		{
			Press("YES");
		}
		else if (CurrentMessage() == -1)
		{
			Press("AMEND");
		}

		if (CurrentMessage() != -1)
		{
			Abort();
		}

		// Enter the rest of the data onto the form and press OK
		Enter("Address.AddressLine1",	0, ADDRESS_LINE2,	SKIP_IF_NO_DATA);
		Enter("Address.AddressLine2",	0, ADDRESS_LINE3,	SKIP_IF_NO_DATA);
		Enter("Address.AddressLine3",	0, ADDRESS_LINE4,	SKIP_IF_NO_DATA);
		Enter("Address.AddressLine4",	0, ADDRESS_LINE5,	SKIP_IF_NO_DATA);
		Enter("Address.AddressLine5",	0, TOWN_CITY_1,	SKIP_IF_NO_DATA);
		Enter("Address.Comment",		0, COMMENT,	SKIP_IF_NO_DATA);
		Enter("Address.Country",		0, COUNTRY,	SKIP_IF_NO_DATA);
		Enter("Address.CountryCode",	0, COUNTRY_CODE,	SKIP_IF_NO_DATA);
		Enter("Address.EDIProfile.EdiProfile_Descr", 0, 264, SKIP_IF_NO_DATA);
		Enter("Address.EDIProfile.EdiProfile_EdiProtocol", 0, 266,
SKIP_IF_NO_DATA);
		Enter("Address.TemporaryAddress", 0, 237, SKIP_IF_NO_DATA);
		Enter("Address.LanguageCode",	0, LANGUAGE_CODE,
SKIP_IF_NO_DATA);
		Enter("Address.LinkAddressCode", 0, LINK_ADDRESS_CODE,
SKIP_IF_NO_DATA);
		Enter("Address.LookupCode",		0, LOOKUP_CODE,	SKIP_IF_NO_DATA);
		Enter("Address.PostalCode",		0, POSTAL_CODE,	SKIP_IF_NO_DATA);
		Enter("Address.ShortHeading",		0, SHORT_HEADING,
SKIP_IF_NO_DATA);
		// Short Heading must be entered
		if (CurrentMessage() != -1)
		{
			Abort();
		}
		Enter("Address.State",		0, STATE,	SKIP_IF_NO_DATA);
		Enter("Address.StateCode",	0, STATE_CODE,	SKIP_IF_NO_DATA);
		Enter("Address.TelephoneNumber",	0, TELEPHONE_NUMBER,
SKIP_IF_NO_DATA);
```

```
              Enter("Address.FaxNumber",   0, TELEX_FAX_NUMBER,    SKIP_IF_NO_DATA);
              Enter("Address.TimeZone",     0, TIME_ZONE,    SKIP_IF_NO_DATA);
              Enter("Address.TownCity",     0, TOWN_CITY_2,    SKIP_IF_NO_DATA);
              Enter("Address.WebPageAddress",  0, WEB_PAGE_ADDRESS,
SKIP_IF_NO_DATA);
              Enter("Address.Area",         0, AREA,    SKIP_IF_NO_DATA);
```

The first thing it does is to load the Address function within SunSystems 5:-

```
OpenFunction("VW5138", "SASI");
```

VW5138 is the internal code for this function, and SASI is the name of the form to use.

SASI uses special forms which contain every field available for that function. They are not formatted in any way as they are not seen by the user.

The next line tells the routine to loop round and import the data, whilst there is still data coming in from the input payload:

```
while (DataIsAvailable("Address"))
```

The following lines start to enter the information on the screen, populating text fields, and pressing buttons:

```
Enter("Address.AddressCode", 0, ADDRESS_CODE, ERROR_IF_NO_DATA);
if (CurrentMessage() != -1)
{
      Abort();
}
Press("OK");
      if (CurrentMessage() == DOES_NOT_EXIST)
{
      Press("YES");
}
else if (CurrentMessage() == -1)
{
      Press("AMEND");
}

if (CurrentMessage() != -1)
     {
                Abort();
}
Enter("Address.AddressLine1",    0, ADDRESS_LINE2,    SKIP_IF_NO_DATA);
Enter("Address.AddressLine2",    0, ADDRESS_LINE3,    SKIP_IF_NO_DATA);
```

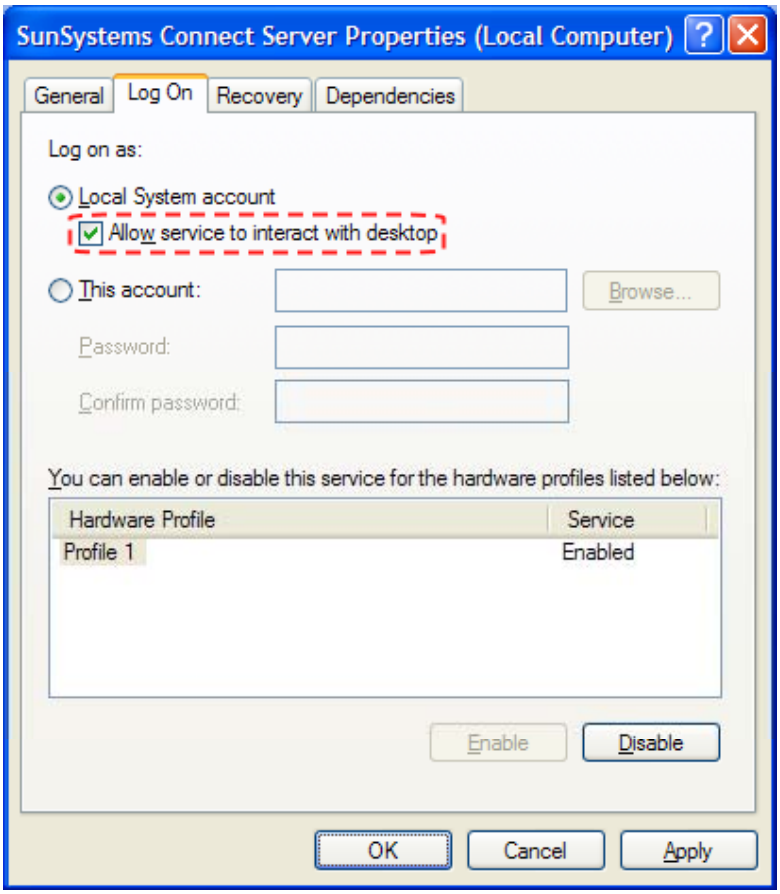
Views

Whilst it is generally accepted that SASI runs in the background so you can't see what is happening, for the purposes of troubleshooting you can temporarily turn these views on, on the application server if you need to see what is happening. This can be useful to see what field a particular error is happening on if the message being returned is not helpful or ambiguous (such as "an analysis code" being incorrect – you can use this to see which one).

In order to do this, you must be on the SunSystems application server, have administrative rights to the SunSystems Services, and the SunSystems Connect service must be running as Local System user account. Therefore it is not possible if the ServerFiles folder is on another machine and the Connect service is running with a specific domain user ID, to view SASI views.

Instructions for switching on the views are as follows:

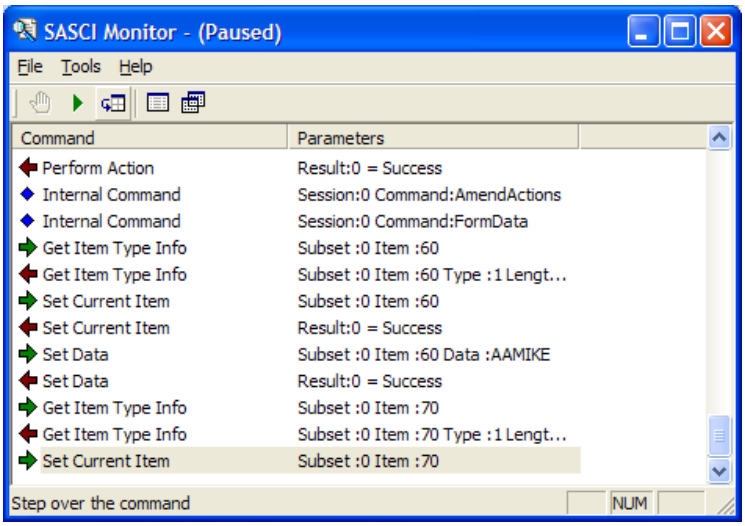
1. Go to Control Panel > Administrative Tools > Services.
2. Stop the SunSystems Connect Service.
3. Navigate to your <SunSystems5>\ssc\bin directory.
4. Double-click “Enable SASI Views.reg” and confirm all messages that appear.
5. Go back to Services, right click the SunSystems Connect Service and select “Properties”.
6. Click the Log on tab, and put a tick in the box marked “Allow service to interact with desktop” as shown:



7. Apply the change and re-start the Connect Service.

   Note: If you are using a domain user account to access the ServerFiles on another computer, you will not be able to use the “Allow service to interact with desktop” option. In this case you may wish to temporarily move the ServerFiles back to your SunSystems application server in order to use this functionality. This will involve then editing the SunSystems Domain Database to point to the new location of the ServerFiles.

When you next run a SASI import, you will see a small window pop up in the lower-right corner of the screen. This is called the SASCI monitor, it contains a log of all the commands sent to and received from SunSystems:



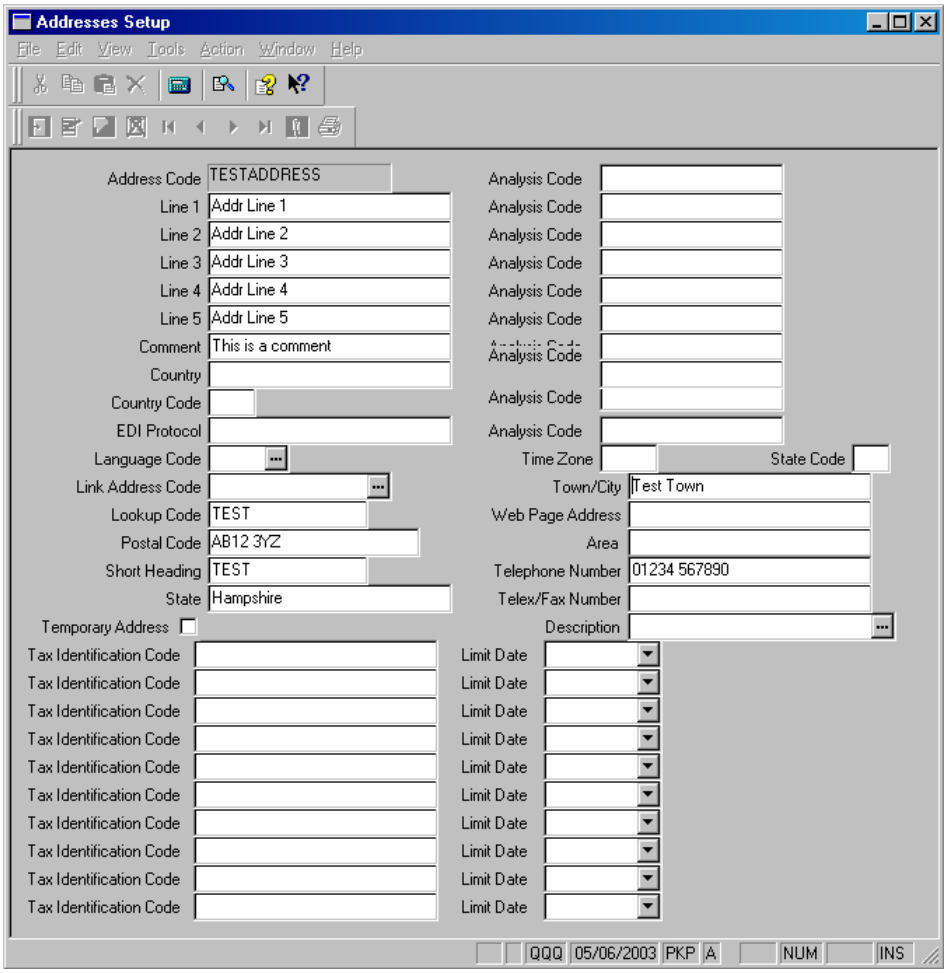
Additonally the SASI forms will appear on the screen of the application server, so you can see the fields being populated one-by-one.

This process happens very fast, and can be difficult for the eye to see what's being populated with what. It is a good idea, once the import starts to press the 'pause' button on the SASCI monitor:

This will stop the current process, and you can use the 'step' button to progress one entry at a time.

Pressing the green arrowed 'play' button, resumes full speed processing.

Here is an example of a SASI form. You can see that it is not designed for user-usage, and is basically a form with every single field for that function, on it:



It is highly recommended that once finished with SASI views, they are switched off again.

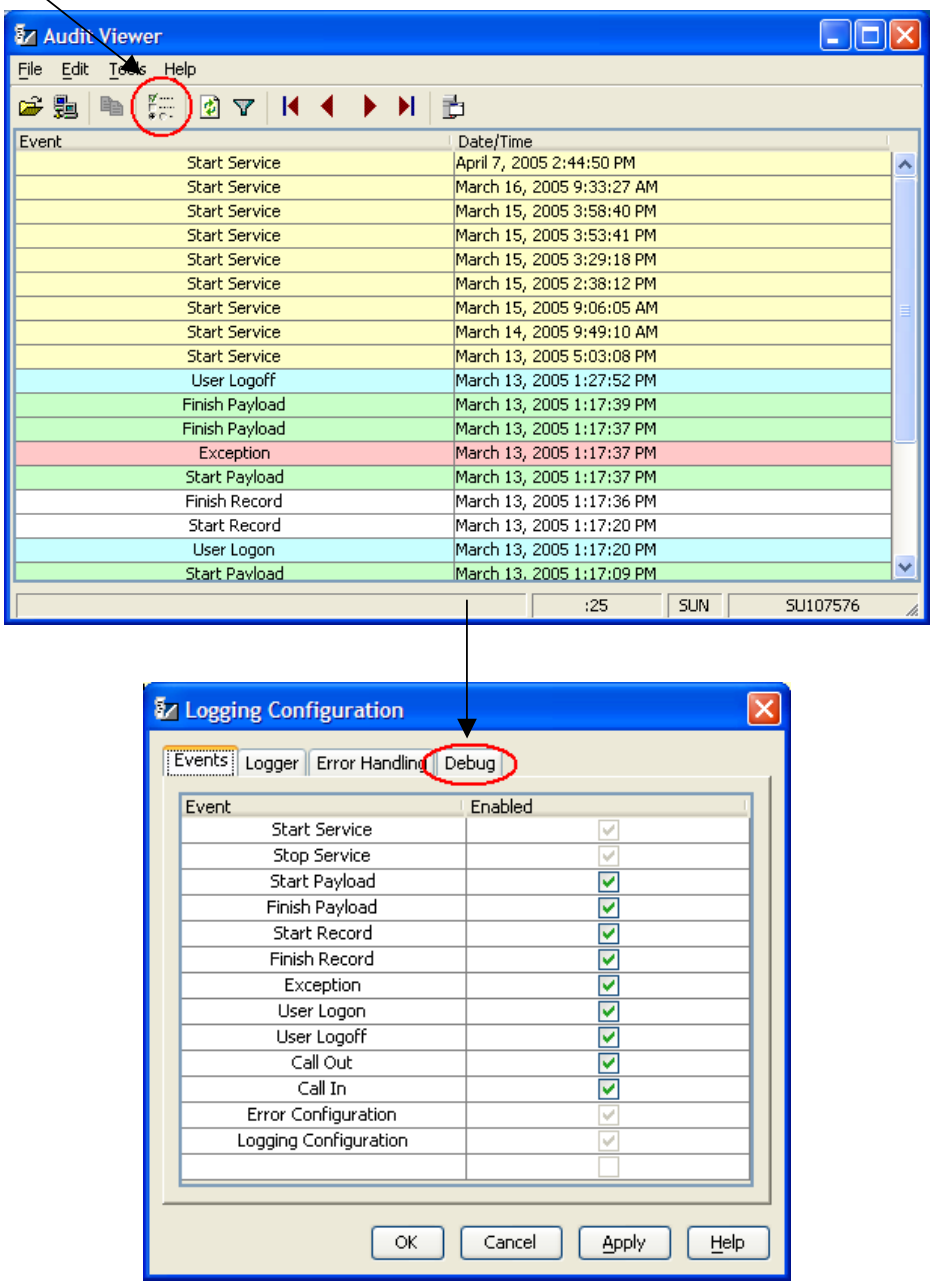
Instructions for switching off the views are as follows:

1. Close the SASCI Monitor.
2. Navigate to your <SunSystems5>\ssc\bin directory.
3. Double-click "***Disable*** SASI Views.reg" and confirm all messages that appear.
4. Go to Control Panel > Administrative Tools > Services.
5. Right click the SunSystems Connect Service and select "Properties".
6. Click the Log on tab, and ***UNtick*** the box marked "Allow service to interact with desktop".
7. Apply the change and re-start the Connect Service.

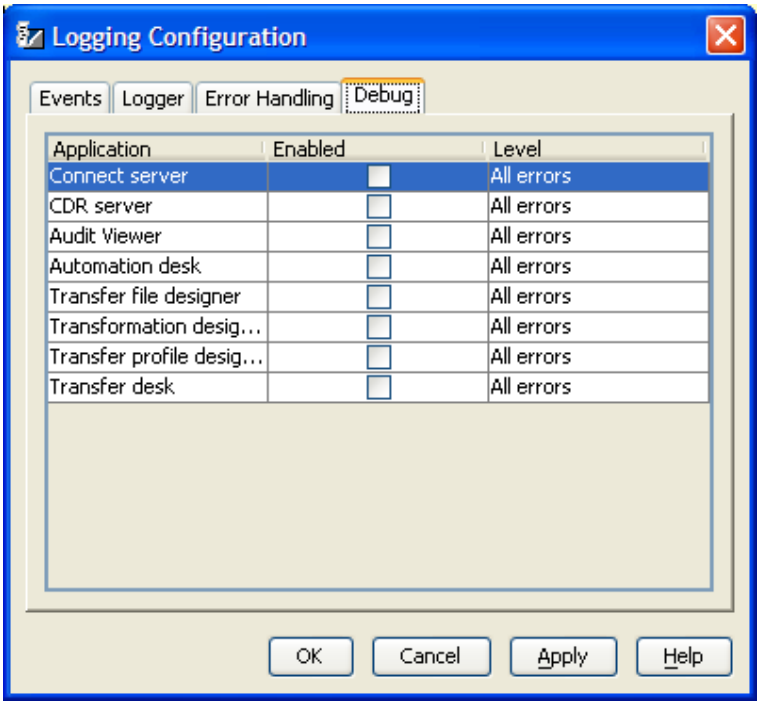
Simple logging via Audit Viewer

As previously mentioned, Audit Viewer has the functionality to enable in-depth logging of SSC and related functions within SunSystems 5.

To enable this, log in to SunSystems as the Sun Superuser, and load AV=Audit Viewer. Click on the Configure Settings button, and then click on the "Debug" tab when it is presented to you:



The following screen is presented:



The various SSC relation functions within SunSystems are listed – for troubleshooting SSC we are interested in the first option “Connect server”.

The other options are all useful to aid in debugging other areas, CDR (Common Data Retrieval), and various Transfer Desk applications.

To enable logging for SSC, check the “Enabled” checkbox next to “Connect server”. Then on the right-hand side of the window, select the level of error output you want. In most debugging cases, this would be “All”, as shown:

Finally, click the “Apply” button.

Logging is now enabled for SSC.

Re-run the problem SSC call, and then look to the following file for the results:

- <SunSystems5>\ssc\logging\server.log

This will contain extremely low-level information regarding the internal workings of SSC. It can be hard to read and usually this level of logging would only be necessary if requested by the SunSystems Support team. However from a development perspective it can be useful to see all the SQL statements being executed, the entering of data onto SASI forms, and full stack traces of any errors that occur.

Ledger Import debugging

Unlike the previous error logs which deal with logging from an SSC perspective, where ledger is concerned you can switch on additional internal ledger logging, to aid in the debugging of more data-related issues.

In order to turn this on, browse to the base SunSystems folder on the application server and create a new plain-text file called “LI.INI”. Put the following text into the file:

[LedgerImport]
LI-Generate-Log=Y

Then, after rerunning, there will be a log file with a name of the form LILog_yyyymmdd_hhmmsshh.log (e.g. LILog_20050412_10150614.LOG). Again, this will be located in the base SunSystems installation folder.

```
LIReadDetailRecord   : Import read, LineNo=000000001,Acnt=52100
,AP=000000000,BaseAm=000000000000400.000 ,DC=D,AllocDate=2005-04-12 10:15:04
,ADInd=0000 ,DueDate=                          ,DDInd=0001-
LIValidateImportFile : Userlocator returned data source = SUN521SP1
LIValidateImportFile : Databaselocator returned page code = 1252
LIProcessImportRec   : Acnt=52100          ,Perd=000000000
LIValidateImportRec  : Acnt=52100          ,Perd=000000000
LIValidateImportRec  : Ref File read, KeyValue=LDGPKP
LIValidateImportRec  : Call to ssp_PKP_LISel0152A Prepared OK
LIValidateImportRec  : Call to ssp_PKP_LISel0204A Prepared OK
LIValidateImportRec  : Current Acnt=52100          , Prior Acnt=
LIValidateImportRec  : Ref File read, KeyValue=ACC52100
,AccTy=B,NumEnterAnal=3333322222[1=M,2=O,3=P],AlphaEnterAnal=NNNNN
,PASPAcntType=0000
LICheckAnalysisCodes : CheckTab=NNNNNY     [Analysis Codes with space or 'N'
get removed]
LICheckAnalysisCodes : ACodes=
LICheckAnalysisCodes : Call to ssp_PKP_LISel0270A Prepared OK
LIInsertMsgRecord    : Type=0004 ,Missing TAX             Analysis Code
(Category Id 16) (Ledger Analysis 06)
```

The above is an example of the output generated by Ledger Import logging. This is also low-level logging, but from the financials application point-of-view rather than the technical SSC point-of-view.

Chapter 9 – Workshop

This is simply an opportunity for you to start writing any application you wish, assisted by the trainer, in order to get a good start in developing the applications required for your organisation.

Appendix A: Journal Import Component Documentation

The following methods are available for the Journal component:

Import
This is the main component, and does all the stages of Ledger Import. (1) Loads all the lines into the posting tables, (2) Validates, (3) If correct posts into the Actual Ledger tables.

Load
This method just does stage (1) of the Ledger Import. Loads the data into the posting tables.

LoadAndPost
This method should not be used because it will always post to the Ledger even if the validation fails, so is dangerous to say the least. The Import method should be used instead.

ProcessPostingDetails
This takes the lines already within the posting tables and validates and then posts into the Ledger. (Stages (2) and (3) of the Import method).

Query
Allows you to pull the Ledger transactions from the Ledger tables.

ValidateOnly
The ValidateOnly method can be used to validate your Journal lines before posting them.

Note: When importing data, we recommend the use of the Import method only, as it provides most protection.

The MethodContext Tag

MethodContext Tag Values

Every SSC Journal XML payload that uses the Import function must have the following tags defined under the SSC MethodContext element:

```
<LedgerPostingParameters>
    <JournalType/>
    <DefaultPeriod/>
    <PostingType/>
    <PostProvisional/>
    <PostToHold/>
    <BalancingOptions/>
    <SuspenseAccount/>
    <TransactionAmountAccount/>
    <ReportingAccount/>
    <AllowOverBudget/>
    <AllowPostToSuspended/>
    <AllowBalTran/>
    <SuppressSubstitutedMessages/>
    <LoadOnly/>
    <Description/>
    <LayoutCode/>
</LedgerPostingParameters>
```

The following is a table describing the valid values for these tags.

Tag Name	Allowable Values	Req'd	Example
JournalType	Valid SunSystems journal types	Y	GJ
DefaultPeriod	PPPYYYY format	Y	0052002
PostingType	"0" – Validate only "1" – Post "2" – Post if no errors	Y	0
PostProvisional	"Y" or "N"	Y	Y
PostToHold	"Y" or "N"	Y	Y
BalancingOptions	Blank – Unused (No Options) "JT" – Journal Type "TD" – Transaction Date "RF" – Transaction Reference "T0" to "T9" - Ledger Analysis 1 to 10	Y	T3
SuspenseAccount	Any valid account	Y	9999
TransactionAmountAccount	Any valid account	Y	9999
ReportingAccount	Any valid account	Y	9999
AllowOverBudget	"Y" or "N"	Y	Y
AllowPostToSuspend	"Y" or "N"	Y	Y
AllowBalTran	"Y" or "N"	Y	Y
SupressSubstitutedMessages	"Y" or "N"	Y	Y
LoadOnly	FOR INTERNAL USE		
Description	Free text	N	Invoice for customer xxxx
LayoutCode	Any valid document type for printing the invoice	N	LIALL

MethodContext Tag Descriptions

<AllowBalTran>
This option allows the creation and posting of system generated balancing transactions when the Post Transactions option is set to Post if no Errors, if the imbalance has been caused by a system calculation. For example, if the imbalance is a rounding difference as a result of currency conversion calculations. If this option is set, the balancing adjustment transaction is posted to the Error Suspense Account, Other Account or Reporting Account identified, depending on the currency value being adjusted.

If this option is blank, system generated imbalances are treated as errors which prevents the journal being posted.

<AllowOverBudget>
This option overrides the budget controls and allows transactions to be posted if they exceed the available budget. See How are Transactions Checked Against Available Budget? in SunSystems on-line help.

<AllowPostToSuspended>
If this is set, transactions can post to ledger accounts with a status of Suspended. If not, a transaction posting to a suspended account is treated as an error.

<BalancingOptions>
The determines the level at which the journal transactions must balance.If this level of balancing is required, you can force the transactions to balance by journal type, transaction date, transaction reference, or one of the ten ledger analysis codes.

<DefaultPeriod>
Default Posting Period. The period to which any transactions with an invalid or absent accounting period are posted.

<JournalType>
The type of journals contained in the current import file. All of the journals in the import file must reference this journal type.

<LoadOnly>
FOR INTERNAL USE ONLY

<PostingType>
This determines whether or not the transactions being processed are posted, and whether they can be posted if they contain errors. Three options are available:

- Validate only - if this option is selected the transactions are validated but not posted.
- Post - if this option is selected the transactions are validated, balancing transactions are generated to balance the journal is necessary and the journal is posted.
- Post if no Errors - if this option is selected the transactions are validated and, providing there are no errrors, posted. This can be overridden for a system generated imbalance by the Allow Bal Txns When Post if no Err setting.

<PostProvisional>
If the Provisional Postings option in Ledger Setup (LS) is Optional and this Post Provisional option is set, the transactions are validated and/or posted as provisional transactions. If this option is not set, the transactions are validated and/or posted as permanent transactions.

<PostToHold>
This option validates the journals and stores them as held journals on the Journal Hold file. This provides an additional level of posting control because they then need to be released from hold and posted. If this option is blank, the journals are validated and posted as provisional or permanent postings. See Holding a Journal to Delay Posting in SunSystems on-line help.

<ReportingAccount>
The account code to which a balancing transaction is posted if the second base or reporting currency journal values do not balance, and the amount of the imbalance is greater than the rounding tolerance.

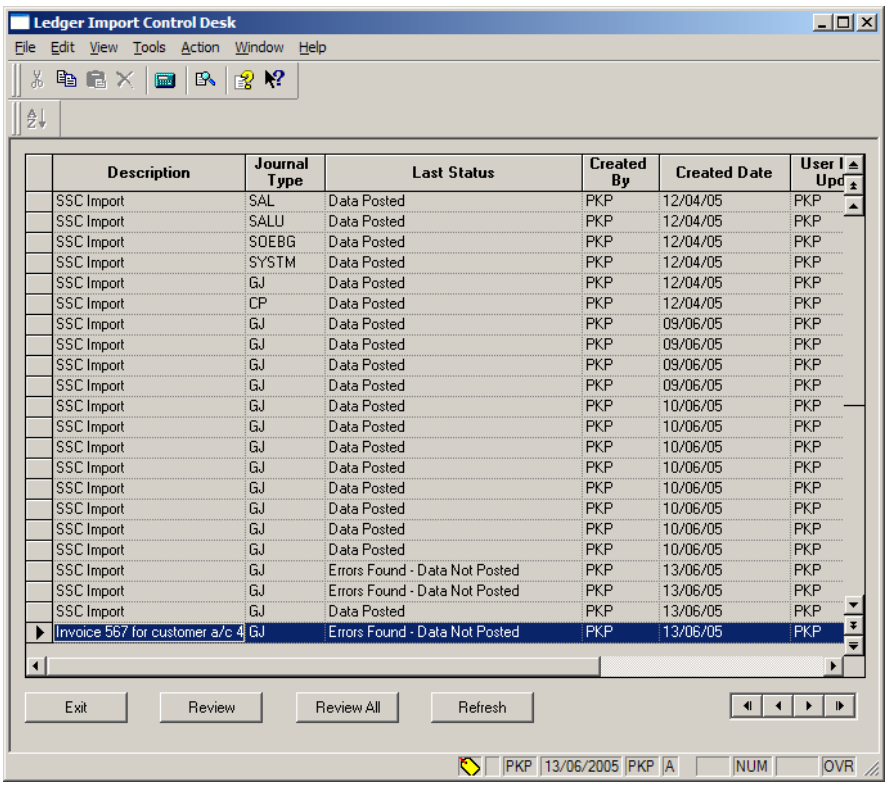
<SupressSubstitutedMessages>
If this option is selected, substitution lines for blank accounting periods and transaction dates, and certain oter low level substitutions, are excluded from the Ledger Import report.

<SuspenseAccount>
The account code to which a balancing transaction is posted if the base currency journal values do not balance. If the account code on a journal transaction is invalid, this account is used as the substitution.

<TransactionAmountAccount>

The account code to which a balancing transaction is posted if the transaction currency journal values do not balance, and the amount of the imbalance is greater than the rounding tolerance.

<Description>
This can be anything you want and it will appear, once imported, as the description shown in LI=Ledger Import. This is to make your imports more identifiable:



Description	Journal Type	Last Status	Created By	Created Date	User I Upd
SSC Import	SAL	Data Posted	PKP	12/04/05	PKP
SSC Import	SALU	Data Posted	PKP	12/04/05	PKP
SSC Import	SOEBG	Data Posted	PKP	12/04/05	PKP
SSC Import	SYSTM	Data Posted	PKP	12/04/05	PKP
SSC Import	GJ	Data Posted	PKP	12/04/05	PKP
SSC Import	CP	Data Posted	PKP	12/04/05	PKP
SSC Import	GJ	Data Posted	PKP	09/06/05	PKP
SSC Import	GJ	Data Posted	PKP	09/06/05	PKP
SSC Import	GJ	Data Posted	PKP	09/06/05	PKP
SSC Import	GJ	Data Posted	PKP	09/06/05	PKP
SSC Import	GJ	Data Posted	PKP	10/06/05	PKP
SSC Import	GJ	Data Posted	PKP	10/06/05	PKP
SSC Import	GJ	Data Posted	PKP	10/06/05	PKP
SSC Import	GJ	Data Posted	PKP	10/06/05	PKP
SSC Import	GJ	Data Posted	PKP	10/06/05	PKP
SSC Import	GJ	Data Posted	PKP	10/06/05	PKP
SSC Import	GJ	Data Posted	PKP	10/06/05	PKP
SSC Import	GJ	Data Posted	PKP	10/06/05	PKP
SSC Import	GJ	Errors Found - Data Not Posted	PKP	13/06/05	PKP
SSC Import	GJ	Errors Found - Data Not Posted	PKP	13/06/05	PKP
SSC Import	GJ	Data Posted	PKP	13/06/05	PKP
Invoice 567 for customer a/c 4	GJ	Errors Found - Data Not Posted	PKP	13/06/05	PKP

If this tag is excluded, the default text “SSC Import” will be used.

<LayoutCode>
This can be any document format code for printing the journal after it has been posted. This is not mandatory.

Appendix B: Security Manager DLL methods

In the case of Win32 applications (written using VB6, C++ 6, etc), you must use SSCClientLibrary.dll to interface to Security Manager. If you are using C++ you can simply include SSCClientLibrary.h in your project.

The methods in this DLL are as follows:

SSCAuthorise

Calls the Connect security manager to verify a username and password. If valid, AUTH_SUCCESS (0) is returned and a voucher string is made available using the SSCReadVoucher method. If the username and password were not valid, or if an error occurs attempting to authorise, a non-zero error code is returned. Additional error information can be obtained using the `SSCGetErrorRespose`, `SSCGetExtendedErrorCode` and `SSCGetExtendedErrorMsg` methods.

```
AuthoriseResult SSCAuthorise(LPCTSTR lpszHostname, int port,
LPCTSTR userName, LPCTSTR password);
```

Parameter	Meaning
`lpszHostname`	Name or IP address of the Connect server that hosts the security manager.
`port`	Port number to talk to Security Manager on. By default, this is 50550. As this can be configured on each server, any application that is not written to target a specific installation should allow this value to be configured.
`userName`	The SunSystems operator ID of the user being authenticated.
`password`	The password for that operator.
`Return value`	A member of the AuthoriseResult enumeration, indicating success or an error code.

SSCReadVoucher

Obtains the voucher returned by the Security Manager. This is only valid once a call to SSCAuthorise has been made, and has returned a success response.

```
LPCTSTR SSCReadVoucher();
```

`Return value`	A voucher string, or NULL if authorisation failed.

SSCGetErrorRespose

Obtains the response message returned by the Security Manager. This is only valid once a call to SSCAuthorise has been made, and has returned a failure response of `AUTH_DENIED` or `AUTH_UNEXPECTED_RESPONSE`.

```
LPCTSTR SSCGetErrorRespose ();
```

`Return value`	A message string or NULL if no message is available

SSCGetExtendedErrorCode

Obtains the error code from a communications error. This is only valid once a call to SSCAuthorise has been made, and has returned a failure response of `AUTH_WINSOCK_ERROR` or `AUTH_CRYPTOAPI_ERROR`.

```
int SSCGetExtendedErrorCode ();
```

`Return value`	A message string or NULL if no message is available

SSCGetExtendedErrorMsg

Obtains the error message from a communications error. This is only valid once a call to SSCAuthorise has been made, and has returned a failure response of `AUTH_WINSOCK_ERROR` or `AUTH_CRYPTOAPI_ERROR`.

```
LPCTSTR SSCGetExtendedErrorMsg ();
```

`Return value`	A message string or NULL if no message is available