akana
by Perforce

SOLA Developer User's Guide Release 6.4.2

Revision Date: August 2017

Contents

About SOLA Developer

The SOLA Developer is a browser based Integrated Development Environment that you can use to create, manage, secure and test services.

The SOLA Developer can be deployed in any standard J2EE container (Tomcat, WebSphere and Weblogic are recommended) and accessed using an internet browser (Microsoft Internet Explorer version 7 or higher is recommended).

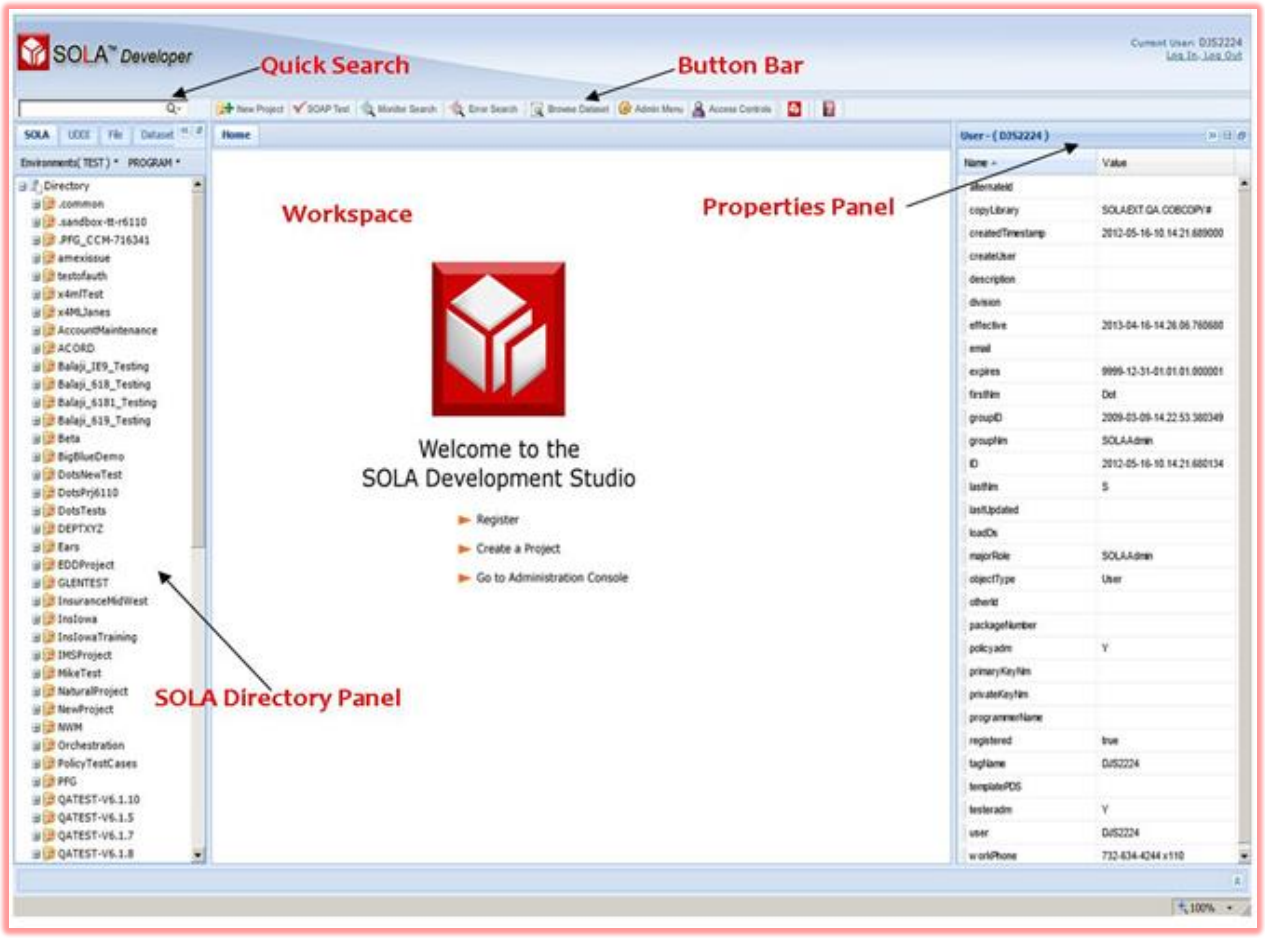
SOLA Developer features a rich graphical interface with advanced Web 2.0 capabilities, such as tabbed workspaces and drag and drop.

SOLA Developer Basics

SOLA Developer is a powerful and easy to use web browser-based development tool with all of the features and functionality of a windows application without the messy install, stringent hardware requirements or resource hogging. Before you begin to learn how to create and govern web services using SOLA Developer, please take a few minutes to become familiar with the development tool and its basic functions.

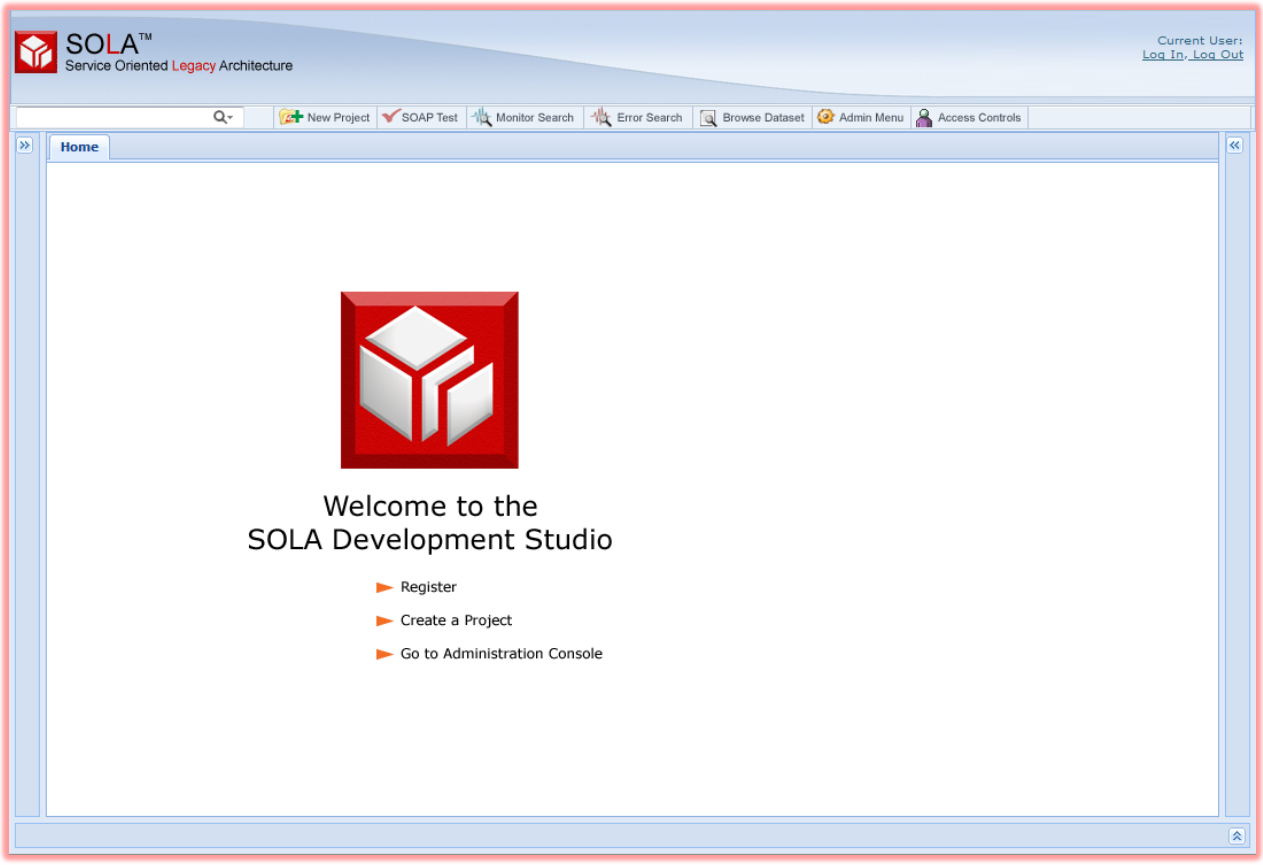
The SOLA Developer Window

The SOLA Developer window is divided into several parts, illustrated in the figure below.

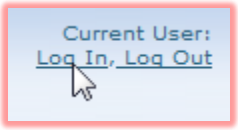


Some of the panels in SOLA Developer can be minimized by using the minimize buttons ( ). When working on lower resolution displays, minimizing panels can clear up work space.

The following illustration shows how much workspace can be cleared up by minimizing all side panels.

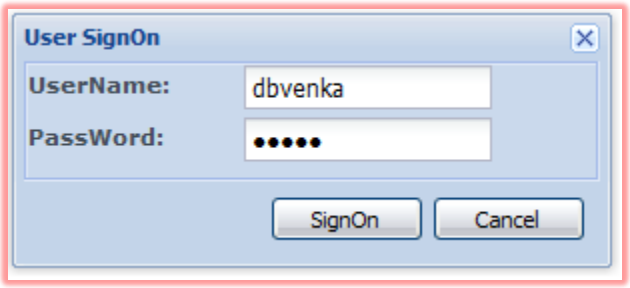


Logging In and User Properties



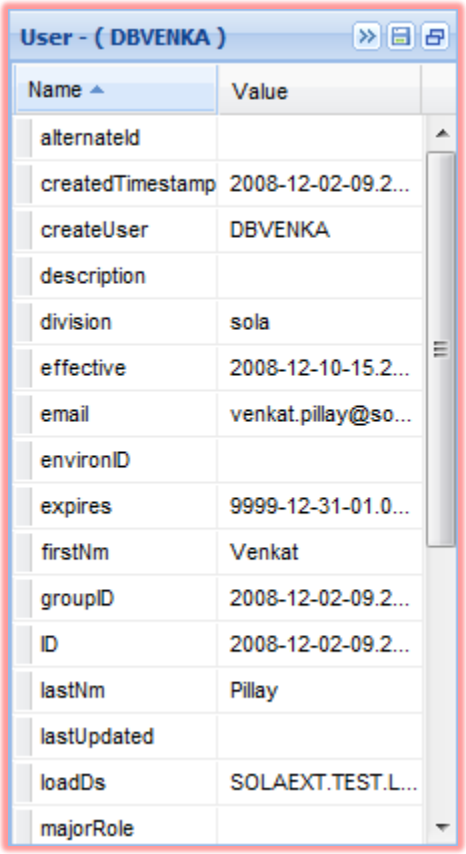
Most of the functionality of SOLA Developer is restricted to authorized users. If you attempt to access a restricted function, you will be automatically prompted to log in. Alternatively, you can log in at any time using the Log In link on the top right of the SOLA Developer screen.

Whether you elect to log in manually or are prompted to do so after attempting to access a restricted feature, you will be presented with a log in prompt.



Enter your TSO username and password and either press the Enter key or click the **SignOn** button. After you have logged in successfully, your username will be displayed above the Log In / Log Out links on the top right of the SOLA Developer window.

Clicking on your name will display your user-level properties in the Properties panel.

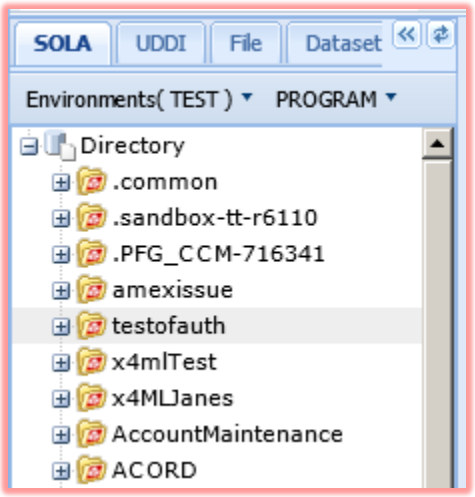


SOLA is highly customizable and so the properties displayed here may not be what you see on your screen. Check with a SOLA Administrator if you have questions about specific properties and their values.

Users can change the values of their user properties by clicking on a value or an empty field in the **Value** column. The value being edited can be either a text field or a drop down menu.

The SOLA Directory Panel

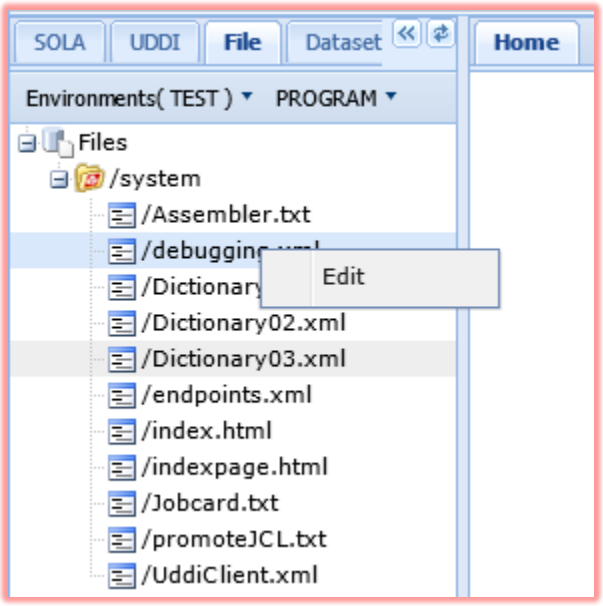
The SOLA Directory is a central UDDI repository that allows for quick and easy discovery, reuse and collaboration. This directory, and all SOLA projects in it, is located in a central location on the SOLA server (mainframe).



The SOLA Directory contains four tabs; SOLA, UDDI, File and Dataset.

SOLA Tab: this tab is the default tab and displays the SOLA UDDI directory where all projects are located containing programs/services and methods/operations..

UDDI Tab: this tab has been deprecated in Version 6.2. It was used to view and interact with third party UDDI directories. UDDI data is now setup within the Admin Menu.



File Tab: this tab has been deprecated in Version 6.2. This tab was used to browse system files, such as Assembler.txt and debugging.xml. You can now edit these files by selecting the Admin Menu tab from the Button Bar. The file will then appear in the workspace.

Dataset Tab: this tab is used to browse the logged in user's datasets. If you are not logged on when you click this tab, you will be presented with a login prompt.

The SOLA Directory functions in a manner very similar to that of Windows Explorer. Projects are represented by folder icons (see legend below) and contain programs, which in turn contain methods. If a directory item such as a project or a program has members (like files in a folder) then that item can be expanded by clicking on the + icon next to it.

To refresh the SOLA Directory click the refresh button ( ).

SOLA Directory Icons

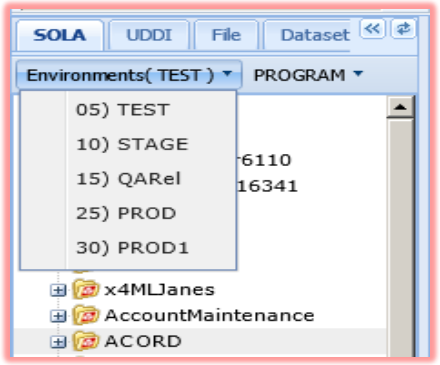
Each program type is represented by a distinct icon. The legend below shows a list of directory items and their associated icons.

- Directory root
- Project
- Commarea program
- IMS Program
- Containers program
- Callable API program
- Outbound web service call program
- BMS3270 – "Green Screen" program
- Adhoc SQL program
- Custom program
- Stored Procedure program
- Orchestrated program

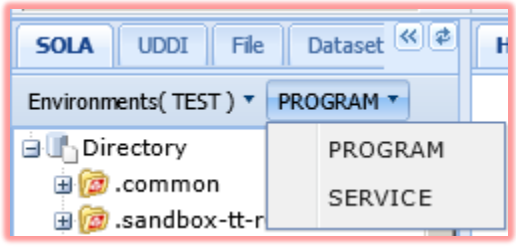
SOLA Directory Filters

The contents of the directory can be filtered by environment and program or service.

Environment names may vary due to SOLA's customizable nature, though SOLA is preconfigured with three environments; TEST, STAGE and PROD.



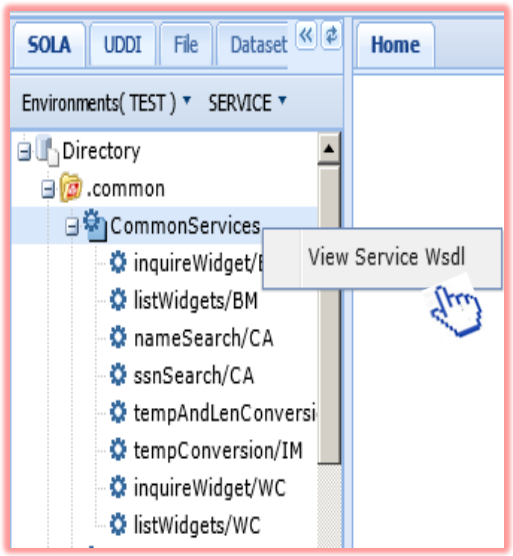
Select the desired environment from the **Environments** menu. Only programs belonging to that environment will be displayed in the tree. For example, if you select the TEST environment, only projects that contain programs in the TEST environment will appear in the SOLA Directory tree.

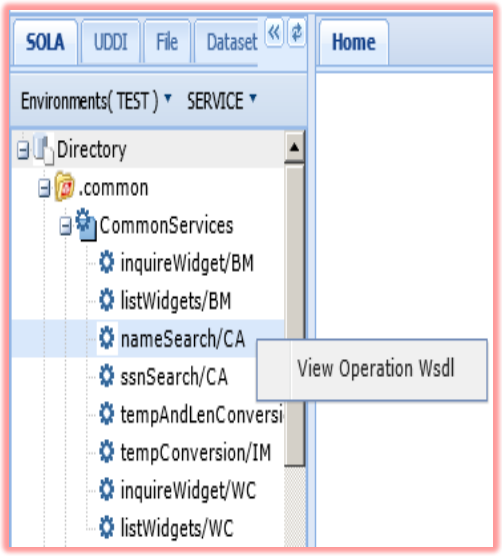


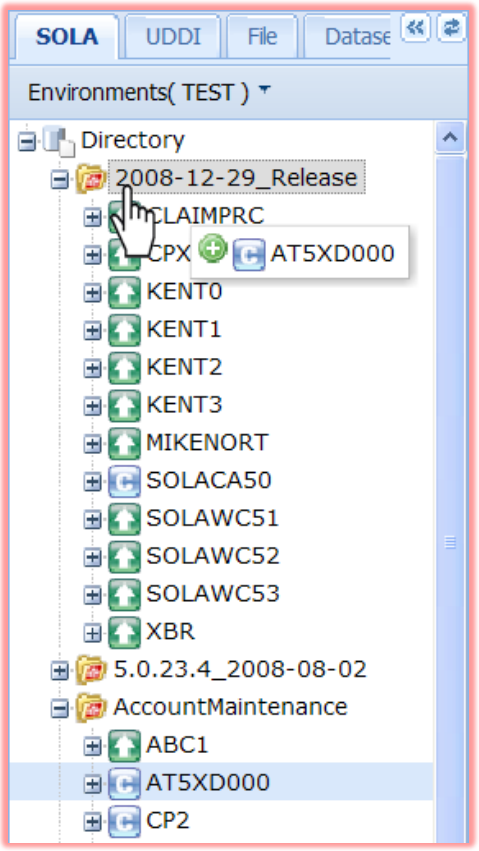
Within an Environment you are able to view the contents of the Directory in either of two modes by selecting one of the following:

Program: displays a list of Projects containing legacy programs and methods. It is in this mode where most development will take place.

Service: displays a list of Projects containing classes referred to here as services, and all of the operations or methods associated with the particular service(s). It is in Service mode that you will have access to the Service WSDL for all operations in that particular Service, and access to the Operation WSDL associated with each operation/method for each service.


SOLA
UDDI
File
Dataset
Home
Environments(TEST)
SERVICE
Directory
.common
CommonServices
inquireWidget/B
listWidgets/BM
nameSearch/CA
ssnSearch/CA
tempAndLenConversi
tempConversion/IM
inquireWidget/WC
listWidgets/WC
View Service Wsdl


SOLA
UDDI
File
Dataset
Home
Environments(TEST)
SERVICE
Directory
.common
CommonServices
inquireWidget/BM
listWidgets/BM
nameSearch/CA
ssnSearch/CA
tempAndLenConversi
tempConversion/IM
inquireWidget/WC
listWidgets/WC
View Operation Wsdl



SOLA Directory Drag and Drop

The SOLA Directory has drag and drop capability which allows you to quickly and easily move programs between projects. A program move from one project to another is governed by following rules:

- User must be a SOLA Administrator or a project administrator for both source and target projects
- The target project must not contain a program of the same name as the one being moved.

To move a program, click on the program, hold down the mouse button and drag the item to its target project.

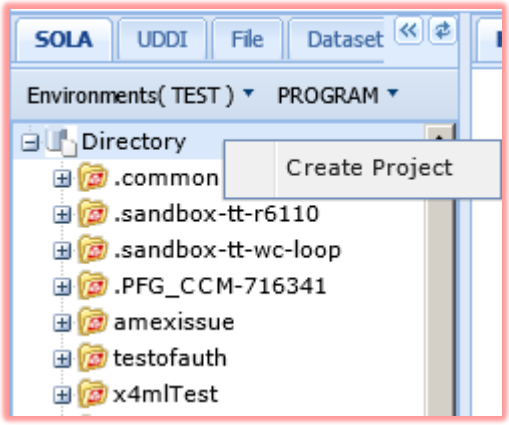
In the example on the left, the Commarea program AT5XD000 is being moved from project AccountMaintenance to project 2008-12-29_Release.

When a program is moved, all of its methods are moved with it from all environments.

SOLA Directory Menus

Right clicking on an item in the directory tree will display a pop-up menu. Many common operations performed in SOLA Developer are initiated using the directory tree menus. Depending on which Directory mode you have selected, PROGRAM or SERVICE, operations will be different.

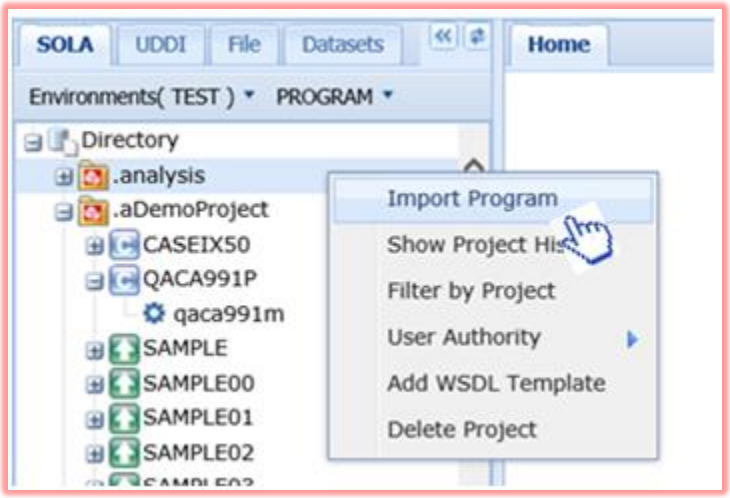
Directory Menu



Create project: right clicking on the **Directory** displays the create project menu and allows you to create a new SOLA Developer project. For more information, see page 22.

Project Menu

The Project Menu is accessed by right clicking on the Project in the Directory tree.



Import Program: displays the import panel, which can be used to import legacy programs and create web services. **The import panel is the gateway to creating web services from every program type that SOLA supports.** This document contains descriptions of the import and analysis process for every supported program. Refer to the table of contents to get information about the program type you are interested in.

Show Project History: displays a list of changes to the project from the day it was created, with date and time stamps for each change. See Figure 1 below:

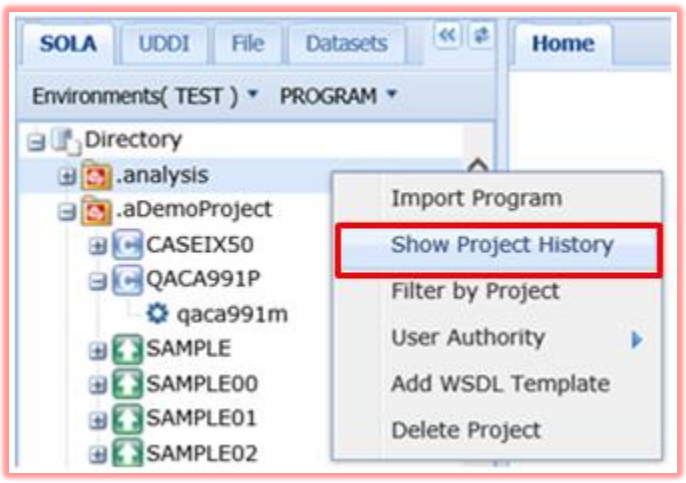
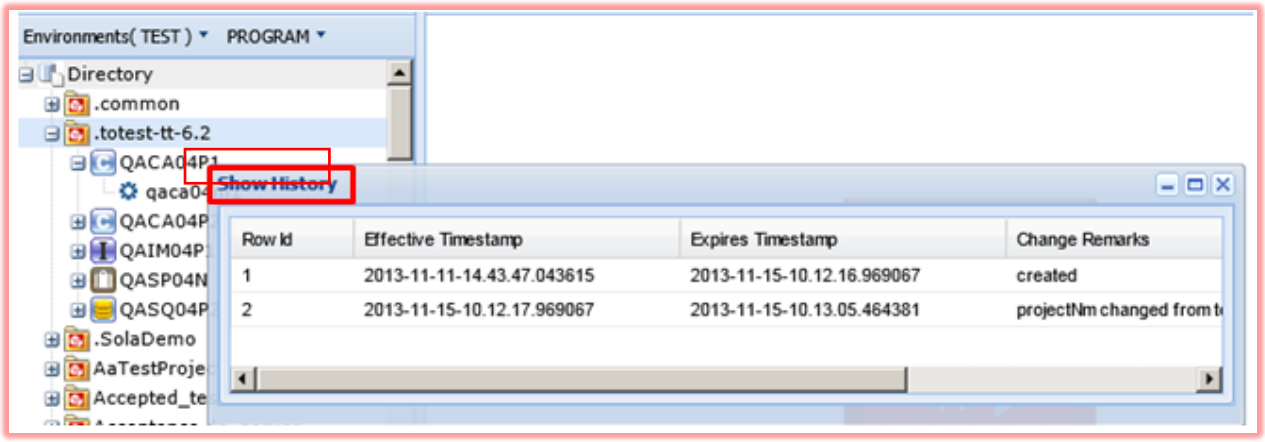
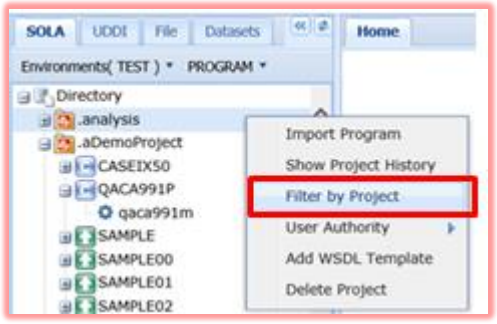


Figure 1:



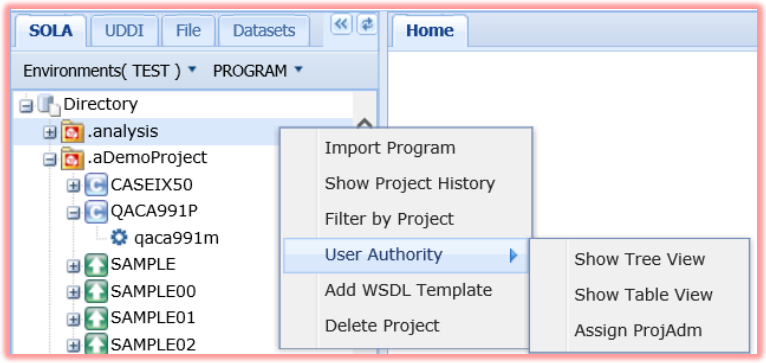
Filter by Project: enables a user to view only a specific project and the programs or services within the project, in the IDE directory tree. A user can share a Project specific URL within a team so developers only see objects within the project when using that URL. An example of this would be:



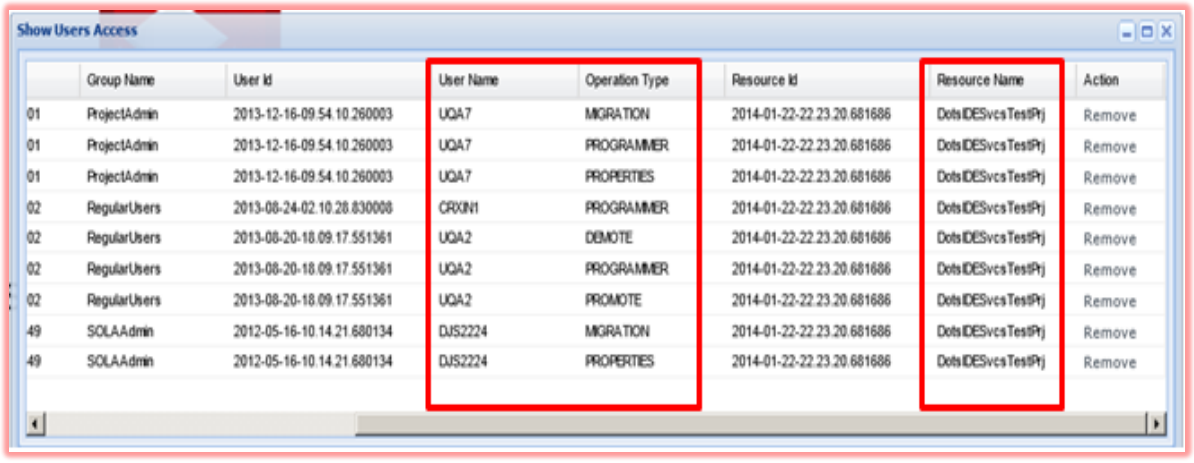
http://<IDEHost>:<IDEPort>/sola/index.html?project=<ProjectName>

Note: To clear the filtering criteria click on the refresh button () as shown below:

User Authority: Enables the management and assignment of the SOLA User's access. Clicking on **User Authority** will display the panel **User Authority Manager** after clicking on the **Show Tree View** option. User Authority functionality is restricted and must be granted by the SOLA Administrator or Project Administrator. For more detailed information about SOLA Developer – User Authority see page 31.



Show Table View: clicking on Show Table View for the Project displays a list of the access **Operation Type** each **User Name** has been assigned for the selected Project/**Resource Name** as seen in the illustration below:

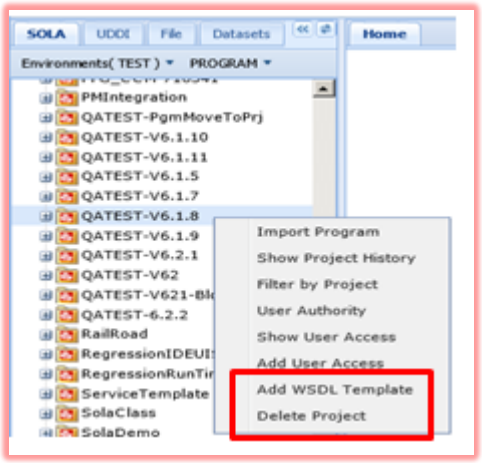


	Group Name	User Id	User Name	Operation Type	Resource Id	Resource Name	Action
01	ProjectAdmin	2013-12-16-09.54.10.260003	UQA7	MIGRATION	2014-01-22-22.23.20.681686	DotsIDESvcsTestPrj	Remove
01	ProjectAdmin	2013-12-16-09.54.10.260003	UQA7	PROGRAMMER	2014-01-22-22.23.20.681686	DotsIDESvcsTestPrj	Remove
01	ProjectAdmin	2013-12-16-09.54.10.260003	UQA7	PROPERTIES	2014-01-22-22.23.20.681686	DotsIDESvcsTestPrj	Remove
02	RegularUsers	2013-08-24-02.10.28.830008	CRXIN1	PROGRAMMER	2014-01-22-22.23.20.681686	DotsIDESvcsTestPrj	Remove
02	RegularUsers	2013-08-20-18.09.17.551361	UQA2	DEMOTE	2014-01-22-22.23.20.681686	DotsIDESvcsTestPrj	Remove
02	RegularUsers	2013-08-20-18.09.17.551361	UQA2	PROGRAMMER	2014-01-22-22.23.20.681686	DotsIDESvcsTestPrj	Remove
02	RegularUsers	2013-08-20-18.09.17.551361	UQA2	PROMOTE	2014-01-22-22.23.20.681686	DotsIDESvcsTestPrj	Remove
49	SOLAAdmin	2012-05-16-10.14.21.680134	DJS2224	MIGRATION	2014-01-22-22.23.20.681686	DotsIDESvcsTestPrj	Remove
49	SOLAAdmin	2012-05-16-10.14.21.680134	DJS2224	PROPERTIES	2014-01-22-22.23.20.681686	DotsIDESvcsTestPrj	Remove

Assign ProjAdm: authorizes a user to work on the project as a Project Administrator. Only the SOLA Administrator can grant this access.

Add WSDL Template: allows users to add a WSDL template containing data defining tags that will be included with all requests and responses sent by programs within the project. This is used to add data tags that are not included when importing a legacy program.

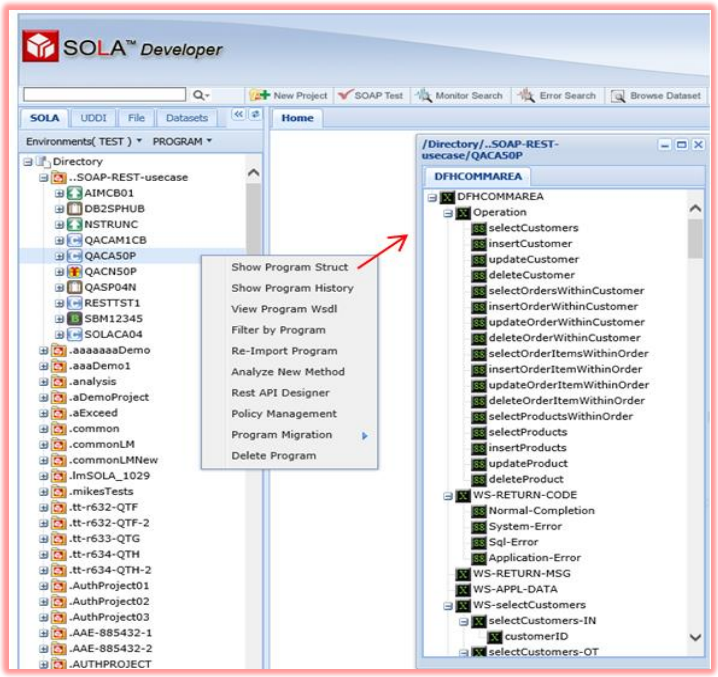
Delete Project: deletes the selected project and all programs and methods in the project. There is no undo function.



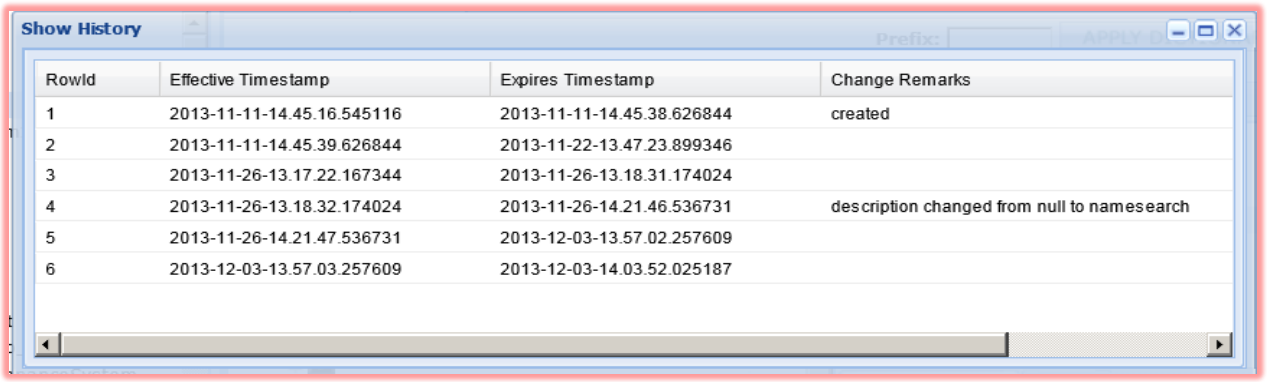
Program Menu

The Program Menu option is a context sensitive menu that varies according to the type of plugin. Each menu option is described as follows:

Show Program Structure: displays the COBOL or PL/I structure of the program (it's interface). For IMS programs, CICS Container type programs and Callable programs with multiple structures, this option shows all of the program's structures using tabs to navigate from one to the other. This option is valid for plugins.

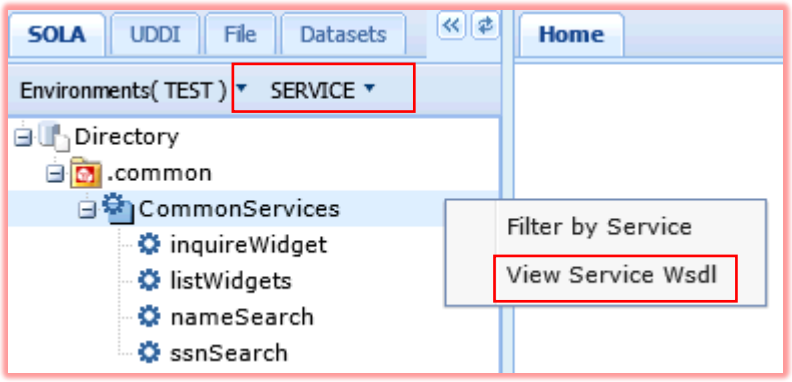


Show Program History: In the environment the user is currently working in, displays a list of changes to the program from the day it was created in the environment, with effective and expires date and time stamps for each change. Right click on **Show Program History** and all history for the program will be displayed including remarks for all changes that have been made to the program. This option is valid for all plugins.



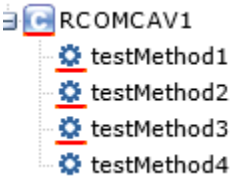
View Program WSDL: Use this option to generate Program WSDL that encapsulates all methods/operations of the program into a single WSDL file. When the directory is in 'Service' View' then you will see 'View Service WSDL' option that consolidates all operations under the service into a single WSDL file. If multiple programs share the same service/class name then service view provides an option to consolidate all operations across multiple programs that share the same service name into a single WSDL file.

This option is valid for plugins.



Filter Program directory will only show the current program on which filter has been applied. To clear the filtering criteria click on the refresh button ( )

Re-Import Program: repeats the import process for the program. For Structure based plugins Re-import is usually done to reflect changes to the structures of a program that has already been captured in SOLA. Depending on the nature of changes to the structure the existing methods/operations that are already captured under the program can become invalid. SOLA automatically verifies the validity of methods based on a Reimport of a program and marks the methods validity. This is visually reflected on the program and invalidated methods with a **'red'** underline as shown below



To fix the methods that have been invalidated, reanalyze the method and adjust the schema items and finalize. Reanalysis is explained in next section as a part of Method Menu documentation. On successful reanalysis of all invalidated methods the Program will automatically change to valid state and the **'red'** underline shown below the Program and each Method will be gone.

As a part of Method Menu, there is a 'Mapping Report' option available that can be used to check the mapping of 'Structure ←→ Input Schema' and 'Output Schema ←→ Structure'. For an invalid method this report gives more details on causes for invalidation. The Mapping report is explained in next section as a part of Method Menu documentation.

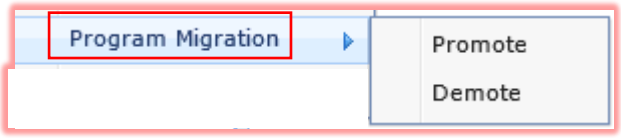
For  plugins, Re-Import will re-capture Outbound, Custom or Stored-Procedure analysis based on the plugin.

Analyze New Method: initiates analysis to create a method for the selected program.

Rest API Designer: Once a legacy Program is imported and a Method analyzed (created) and tested based on the legacy program type, the Rest API Designer presents a single consistent interface within SOLA requiring no additional tools other than command-line http tools such as URI Objects (Resources) and HTTP methods (GET, POST, etc.) to design Representational State Transfer (Rest-ful) API's. See the SOLA Rest User's Guide for further information.

Policy Management: Enables the management and deployment of policies from within SOLA Developer. Clicking on Policy Management will display a Policy Manager panel containing three panes; Containers, Programs and Policies. Policy Management functionality is restricted and must be granted by the SOLA Administrator. For more information about SOLA Developer – Policy Management, see page 24. Policy Management option is valid for all plugins

Program Migration: Has following sub-menu to support Promote/Demote of program



Promote: promotes the program to the next environment in the environment hierarchy and triggers a promotion JCL (if one exists, if not, you can create one using the File Editor). For more information about environments and the environment hierarchy, see page 255.

Demote: demotes the program to the previous environment in the environment and triggers a demotion JCL (if one exists, if not, you can create one using the File Editor). For more information about environments and the environment hierarchy, see page 255.

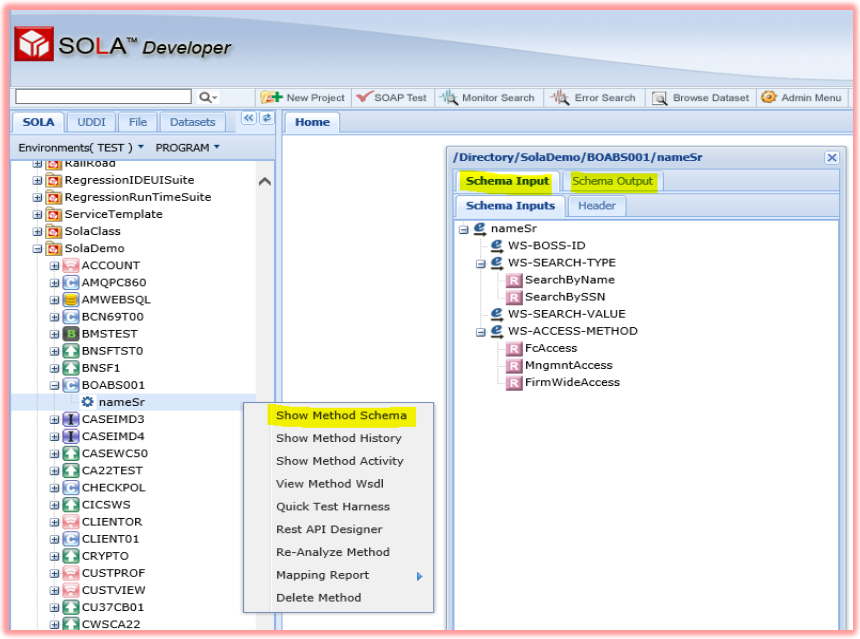
Delete Program: deletes the program and all of the program's methods (if it has any). This cannot be undone.

Method Menu

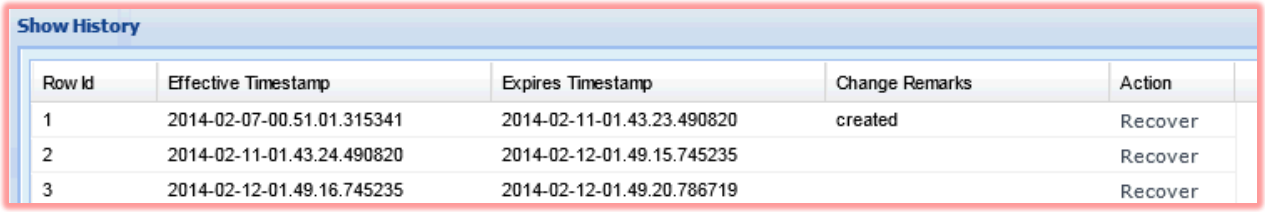
The Method Menu option is a context sensitive menu that is displayed by right clicking on the method / operation name and varies according to the type of plugin.

Each menu option is described as follows:

Show Method Schema: displays the input and output portions of the method's schema (as captured in method analysis). This option is valid for all plugins.



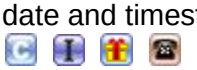
Show Method History: displays a list of changes to the method from the day it was created, with date and time stamps for each change.



Row Id	Effective Timestamp	Expires Timestamp	Change Remarks	Action
1	2014-02-07-00.51.01.315341	2014-02-11-01.43.23.490820	created	Recover
2	2014-02-11-01.43.24.490820	2014-02-12-01.49.15.745235		Recover
3	2014-02-12-01.49.16.745235	2014-02-12-01.49.20.786719		Recover

The above screen has a 'Recover' action that can be clicked on any version to initiate a restore of the method to the selected version. 'Recover' action is only supported for structure based plugins

Note: Method versions that have only captured some changes to method properties without any changes to underlying schema will throw a message 'Method cannot be recovered' as there is no schema items changes that are specific o to that version.

Show Method Activity: displays a list of activity for the method with date and timestamps for each event. This option is only supported for structure based plugins 

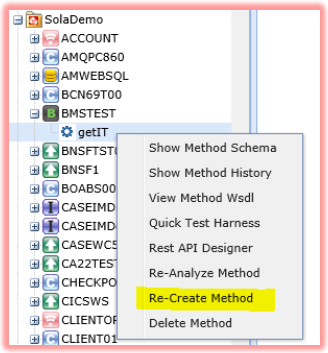
View Method WSDL: Use this option to generate Method specific WSDL. . When the directory is in 'Service' View' then you will see 'View Operation WSDL' option that is exactly identical to method WSDL. This option is valid for plugins.

Quick Test: opens the quick tester panel, which allows you to test the method by sending a request to the legacy program. For more information on testing, see page 218.

Rest API Designer: Choosing the selected Method (previously created during program import and Method analysis) and selecting the Rest API Designer will enable the creation of the REST service. Here you will define the Resources and the HTTP protocol operations that will be described in the SOLA Rest User's Guide.

Re-Analyze Method: repeats the analysis process for the method, this time with all fields pre-populated with their settings from the last analysis. This allows you to make changes to the method by re-analyzing it with different settings, or to view the settings from the previous analysis.

Note: Previously Re-Analysis of methods was allowed in all environments; with release of 6.2 Users are only able to develop services in development/test environments with the lowest Sequence number.



For Custom Plugin( ) methods, 'Re-Analyze Method' option will show already captured method schema to enable users to adjust datatypes, lengths and other attributes that are generated into WSDL. There is a new option 'Re-Create Method' that needs to be selected if the user wants to recapture the method fresh.

For BMS Plugin methods ( ) methods, 'Re-Analyze Method' option will show screens that have been already captured as a part of BMS Method analysis to enable users to adjust the input/output field selection as well as adjusting attributes of screen fields. There is a new option 'Re-Create Method' that needs to be selected if the user wants to recapture the screens fresh.

Re-Create Method: This option is only valid for Custom Plugin((and BMS methods ( ) Use this option to drive the application to recapture the Custom or BMS method fresh.

Delete Method: deletes the method. This cannot be undone.

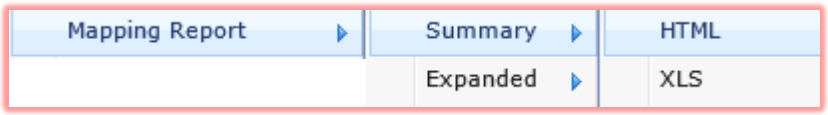
Mapping Report: This option provides user to view the mapping of 'Input Schema → Program Structure' and 'Program Structure → Output Schema' fields.

This option is valid for structure based plugins

Mapping report provides an overview of 2 reports:

- which input schema fields from a soap request are used to populate the programs input structure fields
- which structure fields are used to populate output schema fields of a soap response.

This report has Summary and Expanded reporting options that can be rendered as HTML or XLS format as shown below



The Summary report just captures the mapping field names while Expanded report has the complete xPath information captured for each field so the user knows the location of each field in the structure and schema.

Sample summary report is shown below

Mapping Report ➡ Project : .common Program : RCOMCA04 Method : nameSearch

Table 1: Input Schema Item ➡ Structure Variable Mapping

Input Schema Item	Structure Variable
BossID	WS-BOSS-ID
'N' (default)	WS-SEARCH-TYPE
SearchValue	WS-SEARCH-VALUE
AccessMethod	WS-ACCESS-METHOD

Table 2: Structure Variable ➡ Output Schema Item Mapping

Structure Variable	Output Schema Item
WS-RETURN-CODE	ReturnCode
WS-RETURN-MSG	ReturnMessage
WS-SQL-CODE	SqlCode
WS-CICS-RETURN-CODE	CICSReturnCode

Sample expanded report is shown below

Mapping Report ➡ Project : .common Program : RCOMCA04 Method : nameSearch

Table 1: Input Schema Item ➡ Structure Variable Mapping

Input Schema Item		Structure Variable	
/Schema/Envelope/Body/nameSearch	BossID	/DFHCOMMAREA	WS-BOSS-ID
'N' (default)		/DFHCOMMAREA	WS-SEARCH-TYPE
/Schema/Envelope/Body/nameSearch	SearchValue	/DFHCOMMAREA	WS-SEARCH-VALUE
/Schema/Envelope/Body/nameSearch	AccessMethod	/DFHCOMMAREA	WS-ACCESS-METHOD

Table 2: Structure Variable ➡ Output Schema Item Mapping

Structure Variable		Output Schema Item	
/DFHCOMMAREA	WS-RETURN-CODE	/Schema/Envelope/Body/nameSearchResponse/Dfhcommarea	ReturnCode
/DFHCOMMAREA	WS-RETURN-MSG	/Schema/Envelope/Body/nameSearchResponse/Dfhcommarea	ReturnMessage
/DFHCOMMAREA	WS-SQL-CODE	/Schema/Envelope/Body/nameSearchResponse/Dfhcommarea	SqlCode

When methods are marked as invalid then the mapping report shows the details of why the method got invalidated as shown below.

Mapping Report ➡ Project : .common Program : RCOMCAV1 Method : testMethod1

***** EXCEPTIONS FOUND *****

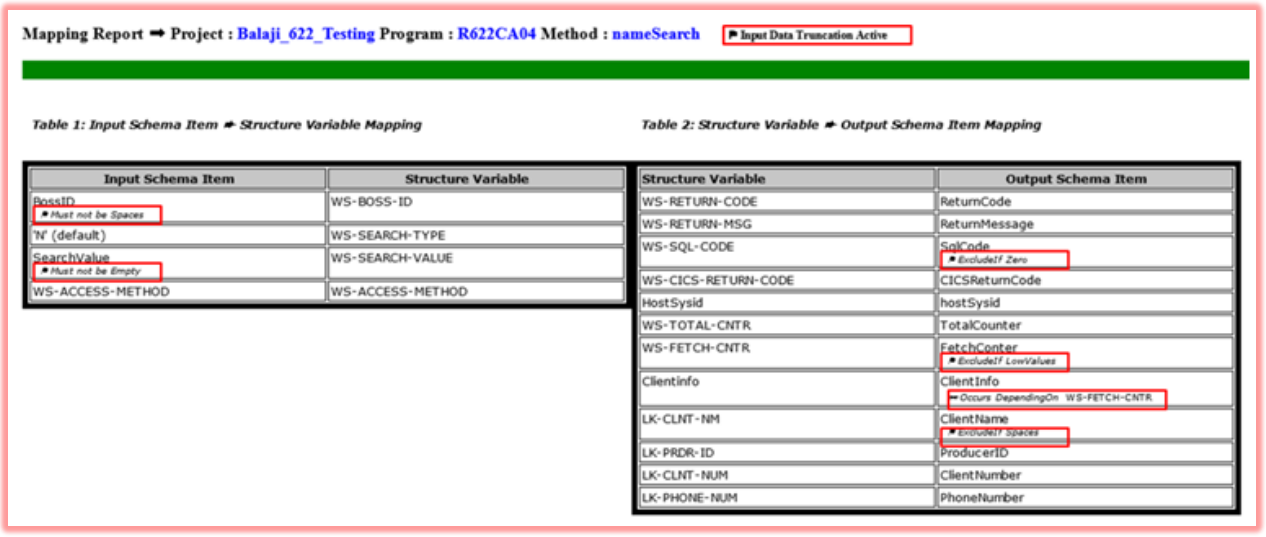
Table 1: Input Schema Item ➡ Structure Variable Mapping

Input Schema Item	Structure Variable
WS-BOSS-ID **** Offset mismatch from this field****	WS-BOSS-ID
WS-SEARCH-TYPE	WS-SEARCH-TYPE
WS-SEARCH-VALUE **** Precision Mismatch ** 25 ⇔ 55**	WS-SEARCH-VALUE
WS-ACCESS-METHOD	WS-ACCESS-METHOD

Table 2: Structure Variable ➡ Output Schema Item Mapping

Structure Variable	Output Schema Item
WS-FETCH-CNTR	WS-FETCH-CNTR
WS-RETURN-CODE	WS-RETURN-CODE
WS-CICS-RETURN-CODE	WS-CICS-RETURN-CODE
LK-CLNT-NM	LK-CLNT-NM
LK-PRDR-ID	LK-PRDR-ID
LK-CLNT-NUM	LK-CLNT-NUM

Color coding (green or red) on report indicates validity of the methods.
Mapping Report also includes Input/Output conditions including the following: AllowTruncation, StopArray, ExcludeIf, OccursDependingOn etc. are outlined below in red.



Mapping Report ➡ Project : Balaji_622_Testing Program : R622CA04 Method : nameSearch ⚑ Input Data Truncation Active

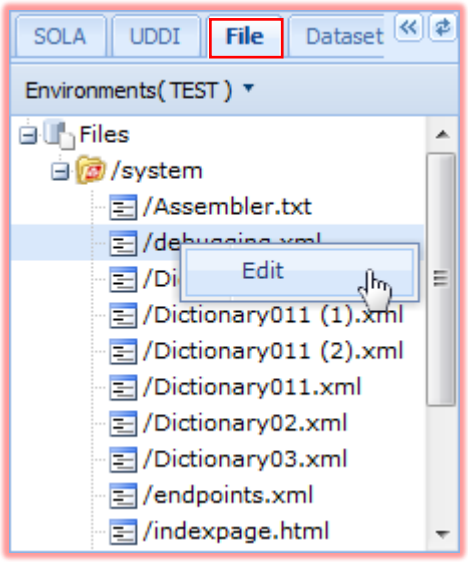
Table 1: Input Schema Item ➨ Structure Variable Mapping

Input Schema Item	Structure Variable
BossID ⚑ Must not be Spaces	WS-BOSS-ID
'N' (default)	WS-SEARCH-TYPE
SearchValue ⚑ Must not be Empty	WS-SEARCH-VALUE
WS-ACCESS-METHOD	WS-ACCESS-METHOD

Table 2: Structure Variable ➨ Output Schema Item Mapping

Structure Variable	Output Schema Item
WS-RETURN-CODE	ReturnCode
WS-RETURN-MSG	ReturnMessage
WS-SQL-CODE	SqlCode ⚑ ExcludeIf Zero
WS-CICS-RETURN-CODE	CICSReturnCode
HostSysid	hostSysid
WS-TOTAL-CNTR	TotalCounter
WS-FETCH-CNTR	FetchConter ⚑ ExcludeIf LowValues
ClientInfo	ClientInfo ⚑ Occurs DependingOn WS-FETCH-CNTR
LK-CLNT-NM	ClientName ⚑ ExcludeIf Spaces
LK-PRDR-ID	ProducerID
LK-CLNT-NUM	ClientNumber
LK-PHONE-NUM	PhoneNumber

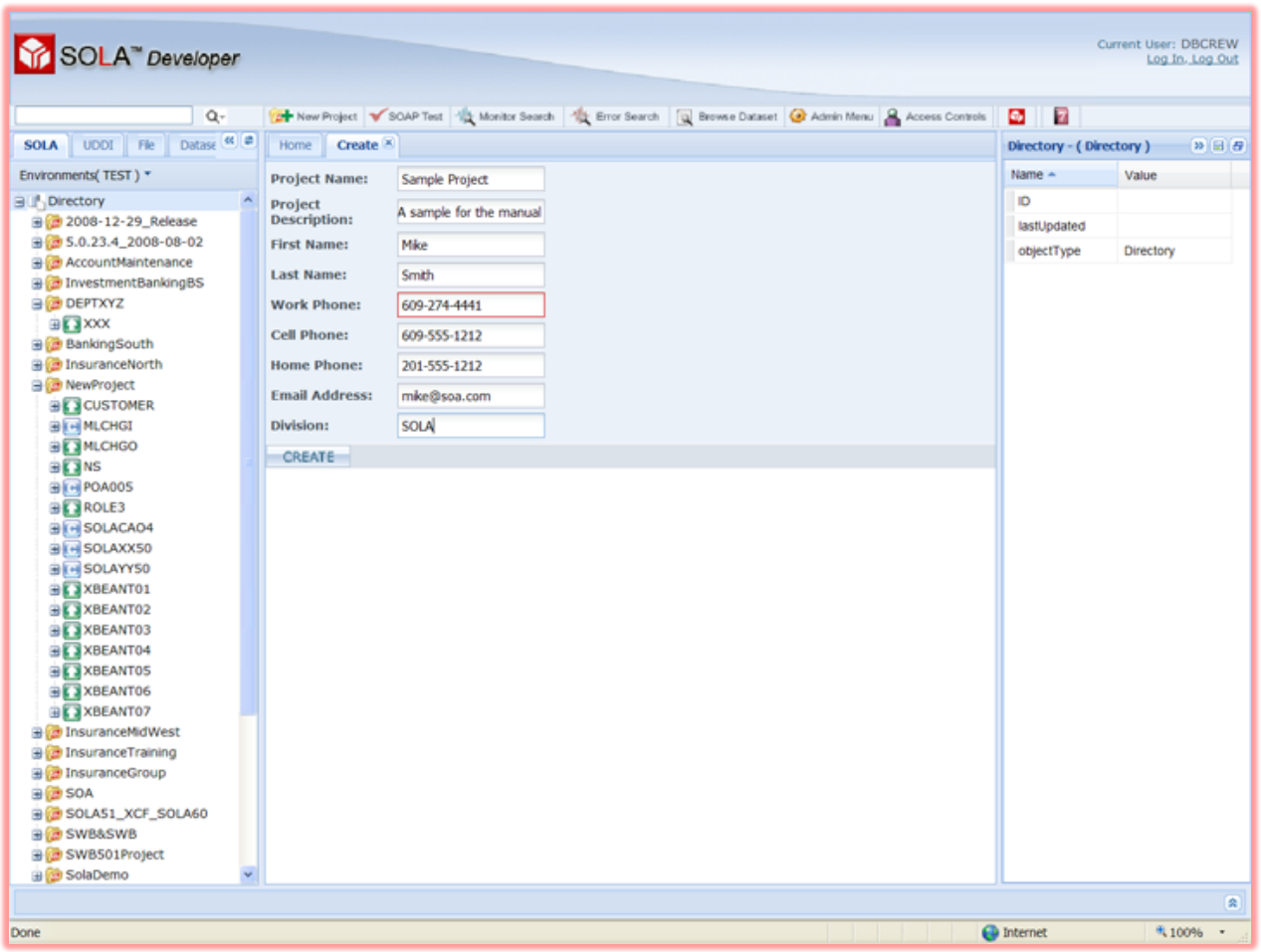
File Tab Menu



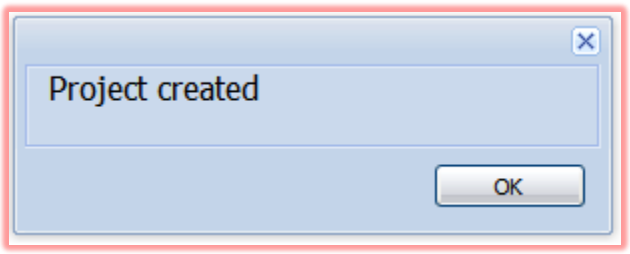
Edit: allows you to edit the selected file in the workspace.

Creating a New Project

To create a new project, either click the **Create Project** button on button bar, or right click the SOLA Directory root icon and select "Create Project". This will display the Create Project panel in the SOLA Developer workspace.

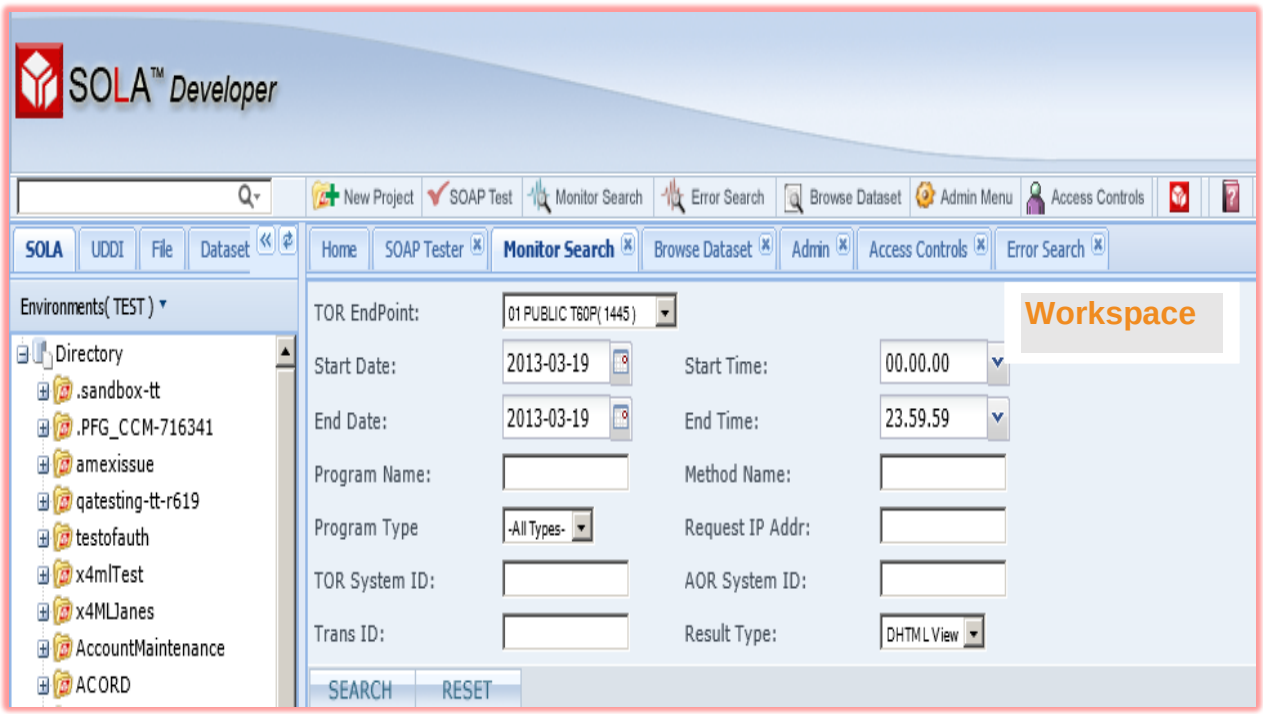


Fill in all required fields and click the **CREATE** button. A confirmation dialog will appear to indicate successful completion.



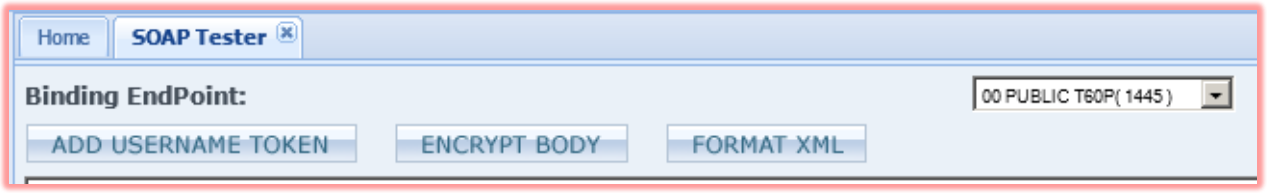
Working with Tabs

The SOLA Developer workspace is tab based; which means that it can contain several active panels, each of which is represented by a tab. The illustration below shows six active tabs in the workspace.



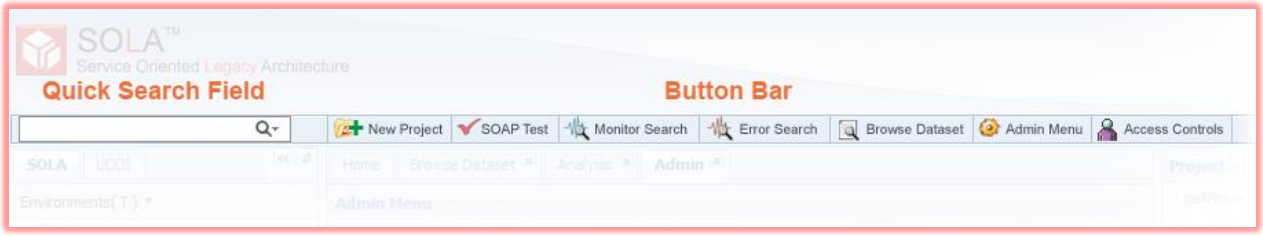
All six can be displayed at once. You can switch between active tabs at any time. This tab based functionality provides several useful benefits, such as the ability to stop working on something, and come back to it later, without having to start from scratch, and the ability to troubleshoot (error search, etc.) without having to abandon what you are working on.

To close a tab, click on the X button in the tab's top right corner.

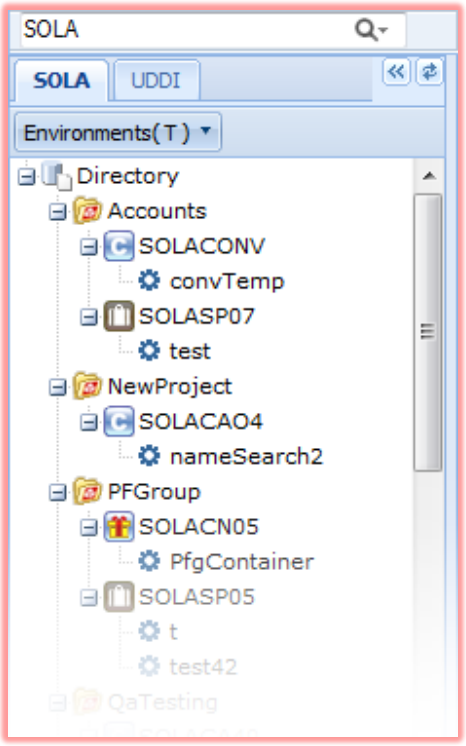


SOLA Developer Toolbar

The SOLA Developer toolbar consists of the quick search field and the toolbar buttons.



Quick Search Field



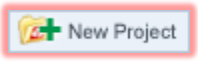
To use the quick search field, type a complete or partial name of the project, program or method you are looking for or perhaps a wildcard character '%' after the program or method, then hit enter. Every item in the SOLA Directory that matches your entry will be displayed. If the matching item is a project or program, that item will be displayed with all child items visible. If the matching item is one or more methods, then only matching methods will be visible in the tree.

To clear the search results and display the full directory tree, click the refresh button ( ).

You can also specify wild card search criteria like %conv%' to search project, program or method that has 'conv' in its name

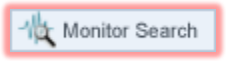
Button Bar

The button bar provides shortcuts and access to some of SOLA's administrative and testing functions.



Click this button to create a new SOLA project. You can also right click on the directory root icon and select "Create Project". See page 22 for details on how to create a project.

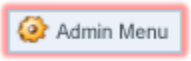
Click this button to access the SOAP tester panel, used to test raw SOAP requests. See page 224 for details on how to use the SOAP tester panel.



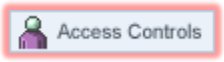
Click this button to access the Monitor Search panel, used to search through all logged SOLA transactions. See page 226 for details on how to use the transaction search panel.

Click this button to access the Error Search panel, used to search through all logged errors. See page 231 for details on how to use the error log.

Click this button to access the Browse Dataset panel, used to view mainframe datasets. See page 235 for details on how to use the Browse Dataset panel.



Click this button to access the SOLA Developer Administration Panel. This panel contains various administration functions related to system files, schemas, dictionary and monitoring. The Admin Panel is detailed on page 238.



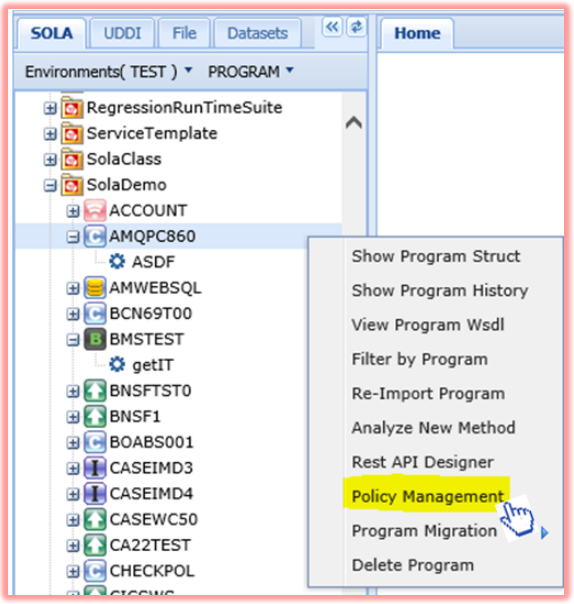
Click this button to access the SOLA Developer User Controls panel. This administration panel contains various functions related to user access. The User Controls panel is detailed on page 257.

Using SOLA Developer - Policy Management

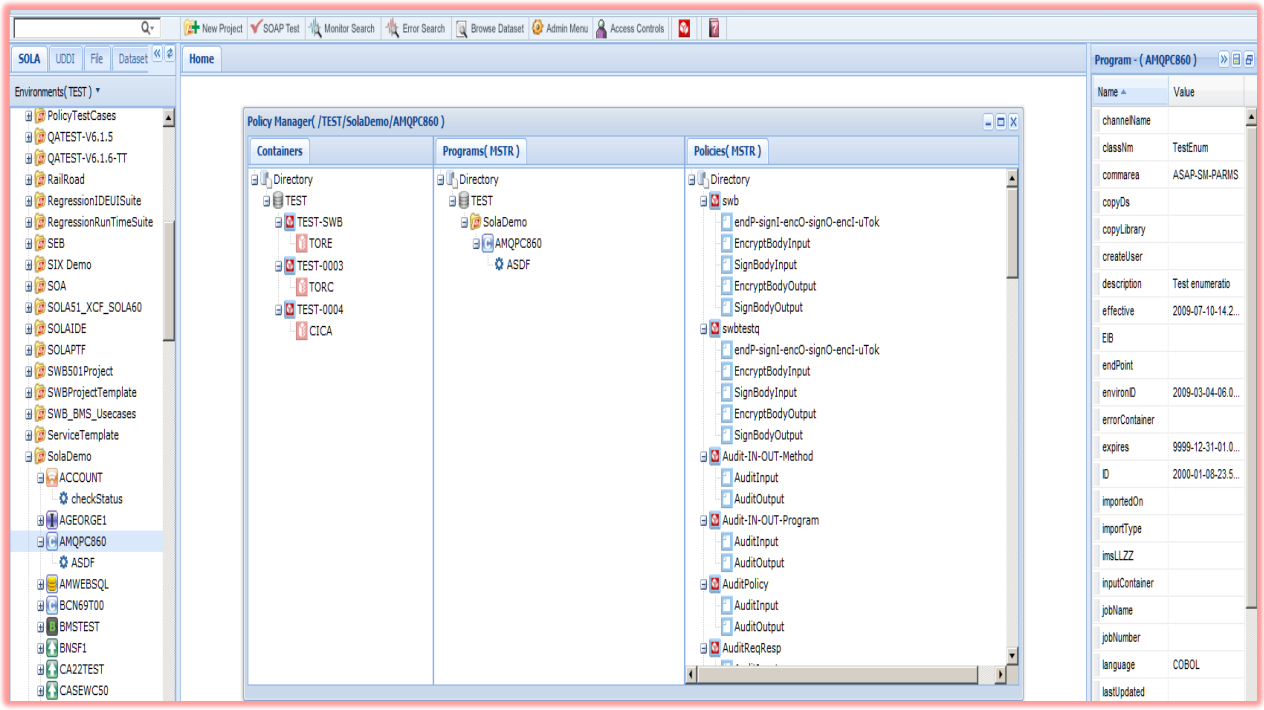
Assigning and Deploying a Policy

The SOLA Administrator must define Policy Management authority to a Project Administrator. This is described in the Resource Manager Users Guide and is granted to a user by granting PolicyAdmin Authority using Resource Manager. A SOLA Administrator serves a dual role as SOLA administrator and Project Administrator; both can assign new user's access to a Project. A user that creates a project is automatically designated a Project Administrator for that project. A Project Administrator has access to project, program and method-level administration features, but cannot see policies on projects they do not have access to.

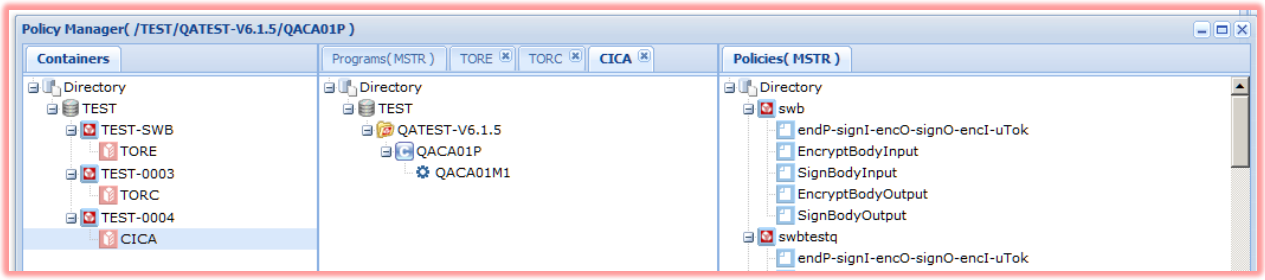
The Project Administrator, if authorized, can assign a policy to a program or method and deploy them into target runtime time containers within an environment. This assignment is accomplished by first right clicking on the program within the project to open the program menu.



After you select **Policy Management** the **Policy Manager** pop-up panel will be displayed in the workspace. This panel can be used to manage policy at the Program or Method level.



The SOLA Developer – Policy Manager Panel contains three panes; Containers, Programs and Policies.

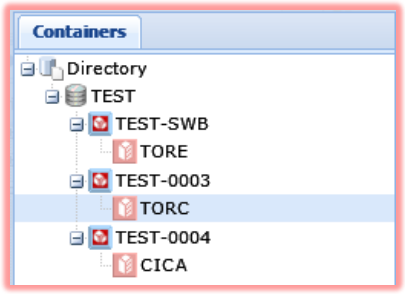


Containers: For ease of reference: Container Groups Containers

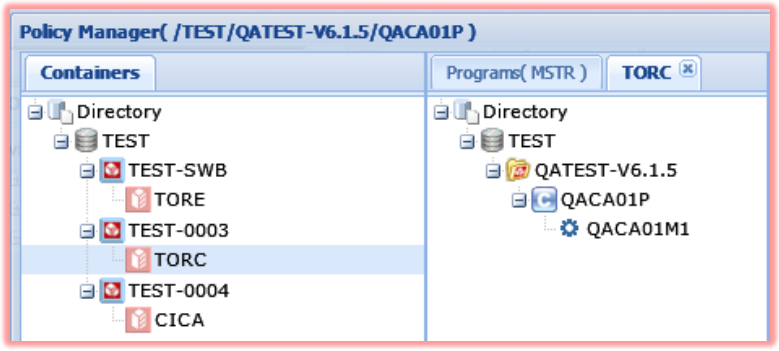
This tab represents the actual SOLA Runtime Containers. The Container Groups and Containers are configured by the SOLA Admin using Resource Manager. Within the Containers Panel is a tree of Container Groups and SOLA Containers represented by container icons (and).

Whenever policies are assigned to a program or method, they are not in effect until they are deployed to a Container Group.

The Directory tree in this example begins with the Environment TEST, followed by three Container Groups defined as TEST-SWB, TEST-0003 and TEST-0004. Each container group in this example has one Container each TORE, TORC and CICA. Containers will store Programs or Methods within Projects (defined with this icon). A Project can be defined by the SOLA Administrator or Project Administrator.



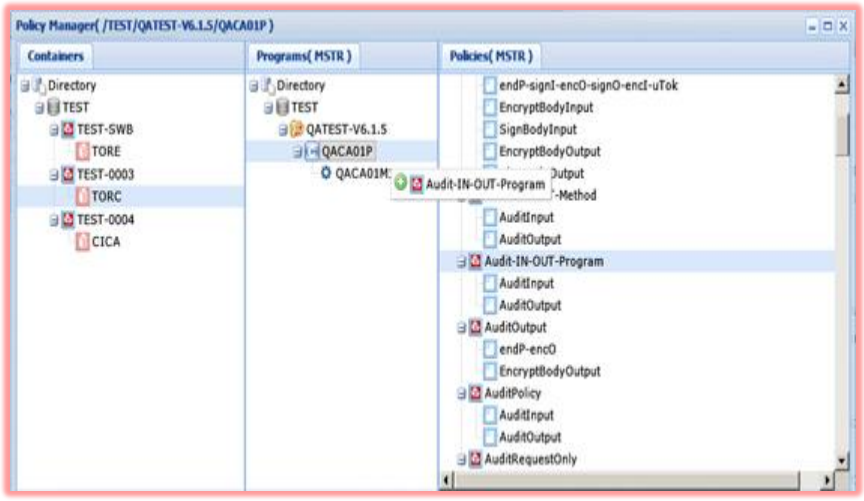
An example of a Container named **TORC** and its contents:



Note: the new TAB that is opened in the **Programs(MSTR)** panel.

Programs or Methods: Are stored within each Project and in a Container(s) . A program or method can be moved to the Containers panel to have its policies deployed in a Container group. All Containers within a Container Group are defined to the same Runtime database (MSTR). When you deploy a program or method to a Container Group it is effectively deployed to all Containers on the Master database.

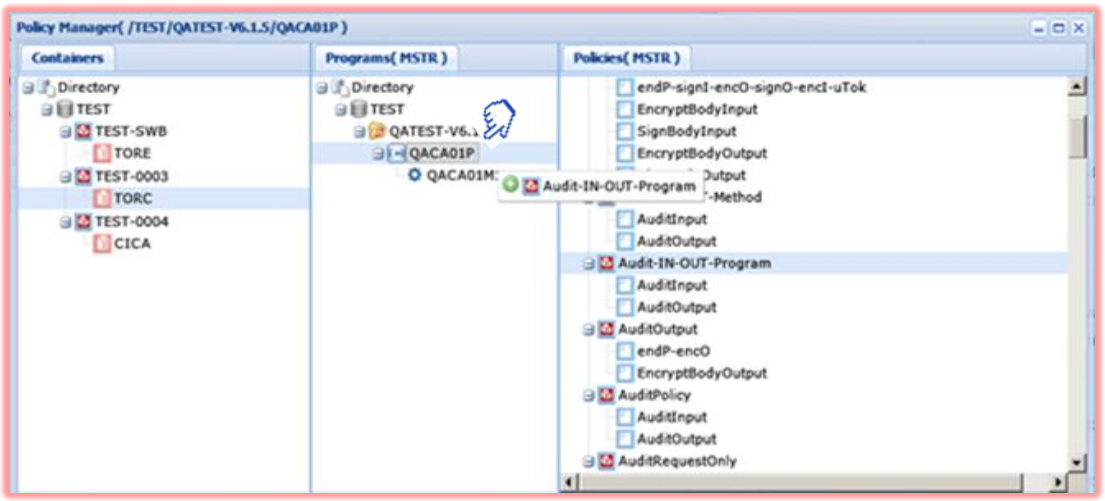
First you must close out of all opened Container TABS TORC in the **Programs(MSTR)** panel by clicking the on each Container TAB, and then drag the program or method to a container group; by doing so the program or method will be deployed and along with its policy(s) activated to every container in that container's group. See the following example:



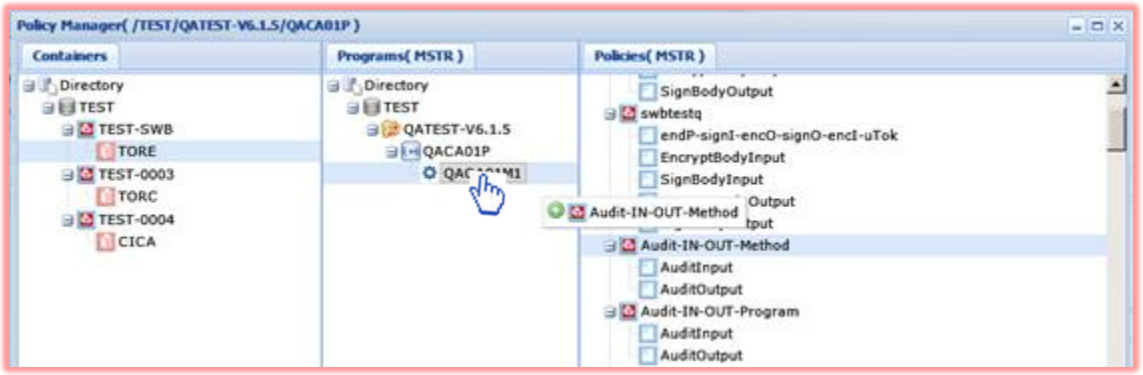
Program QACA01P has been deployed to Container Group TEST-0003, and all Policies in Audit-IN-OUT-Program have been assigned to program QACA01P. The policies now apply not only to the program and/or method, but also to every Container and Container Group within the Runtime database.

Policies: SOLA supports two types of policies, the default policy and the method-specific policy. If a method-specific policy exists, it will always override the default policy. The default policy, which can be enabled or disabled, comes into effect when a method does not have its own policy (and the default policy is enabled).

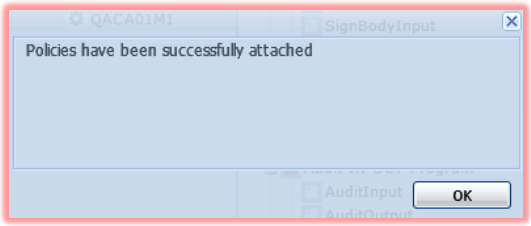
Using Developer's drag and drop capabilities enables a Policy to be dragged over to and dropped into a Program or Method. The policy dropped onto a Program will be assigned to the program and all of its methods as in the example below, the policy group Audit-IN-OUT-Program has been assigned to program QACA01P and all of its methods. Once deployed, the program will use this policy group, overriding the container default policy, except where the default policy defines a requirement set by the assigned policy.



The policy dropped onto a Method will be assigned to the method only as in the example below. Once deployed, the method will use this policy group, overriding the container default policy, except where the default policy defines a requirement set by the assigned policy.

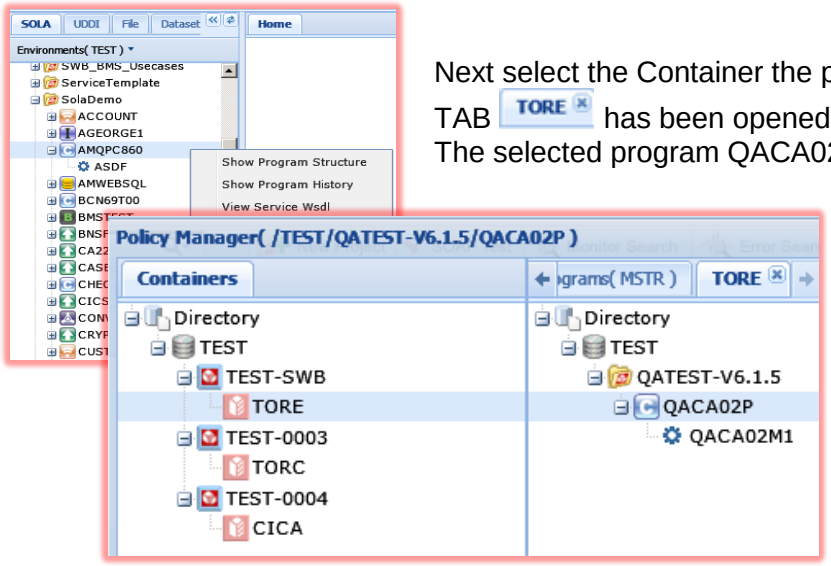


A dialog box will appear confirming the Policy attachment, Click **OK** to continue.



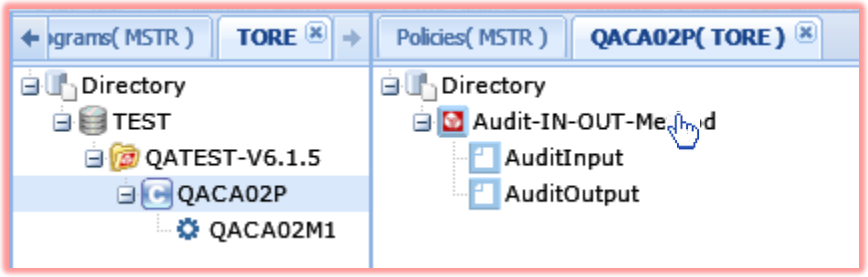
When a Project Administrator has access to projects, programs and methods located in another container group in the Directory tree they will CLICK on the container group and it will appear as a new Icon in the **Programs(MSTR)** panel.

In this example, the project administrator needs to view the current policies assigned to program QACA02P. To do this you would first select the program in the SOLA environment and right CLICK on Policy Management in the drop down menu.

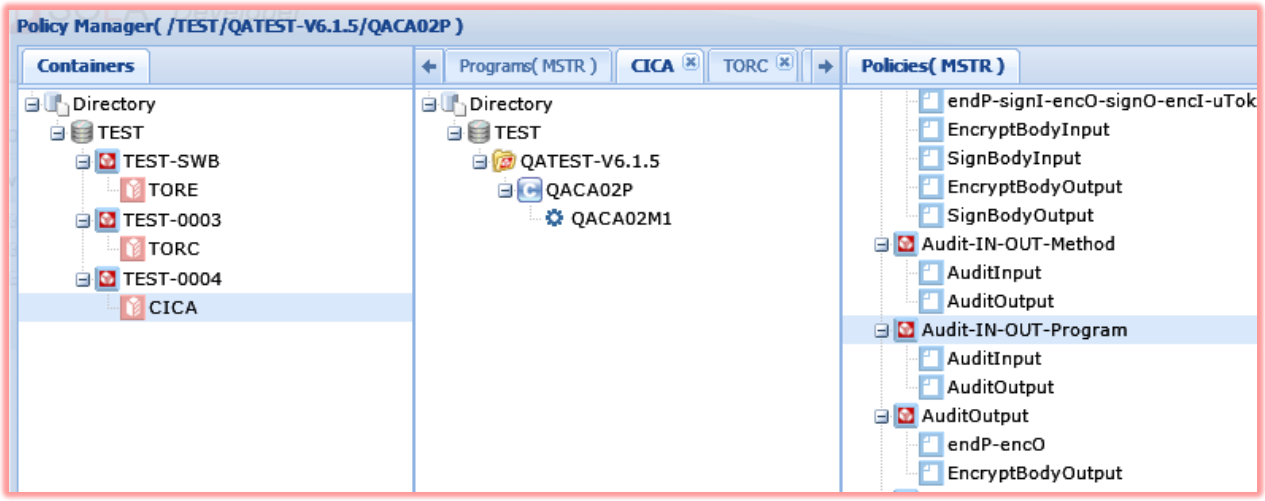


Next select the Container the program is in and notice a new TAB has been opened in the **Programs(MSTR)** panel. The selected program QACA02P is highlighted.

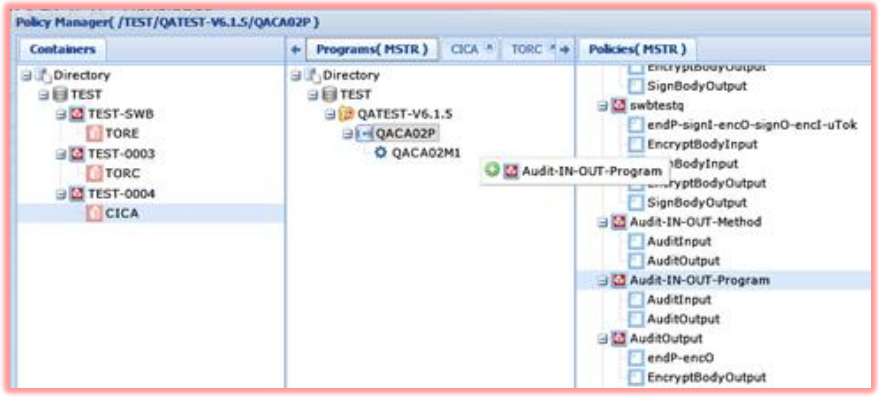
By double-clicking on QACA02P a new TAB will appear in the Policies(MSTR) with the name of the container the program is located in and the current Policy assigned to QACA02P which in this case is policy group Audit-IN-OUT-Method.



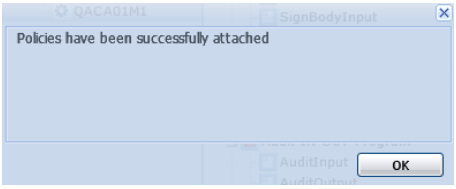
In this example, the project administrator is viewing their projects in container group CICA, and they want to assign policy group Audit-IN-OUT-Program to program QACA02P.



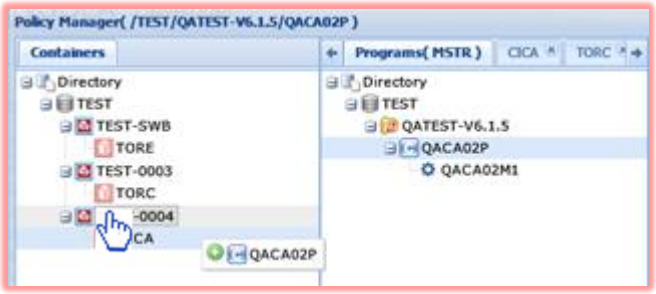
They will have to CLICK on the **Programs(MSTR)** tab to assign the policy to the program by dragging and dropping the policy group onto the program.

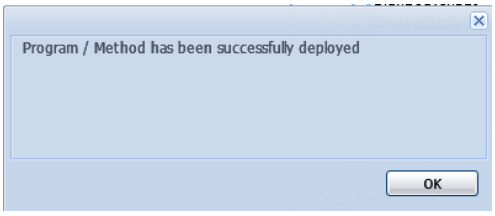


Confirm the policy has been attached by Clicking **OK** to continue.



Then deploy the program by dragging and dropping it onto the target container group:





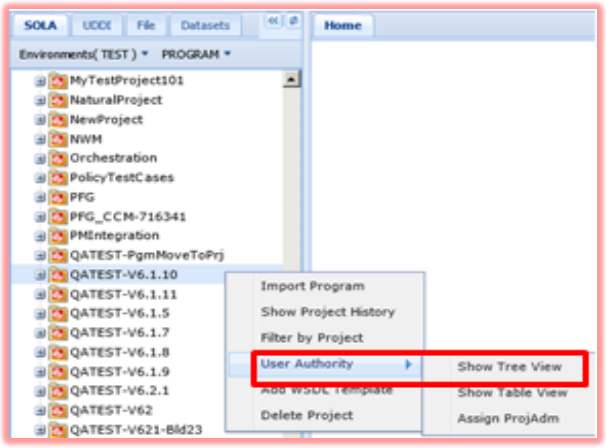
Confirm the policy has been attached by Clicking **OK** to continue.

Using SOLA Developer – User Authority

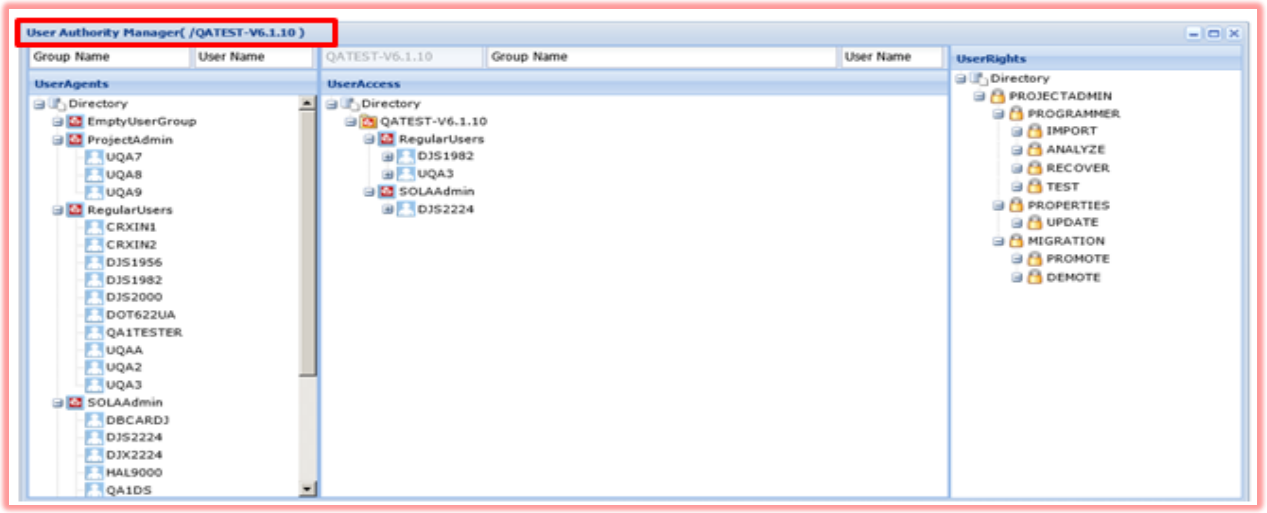
Assigning User Access

To create web services in SOLA a User must first have the proper access authorization. New User access is assigned by the SOLA Administrator or Project Administrator. A SOLA Administrator serves a dual role as SOLA administrator and Project Administrator; both can assign new user's access to a Project. The SOLA Administrator must define user access authority first to a Project Administrator *(this is described further in the Resource Manager Users Guide.*)

A user that creates a project is automatically designated a Project Administrator for that project and the Project Administrator(s) has access to all User Authority for the project. User Authority assignment is accomplished by first right clicking on the Project and selecting **User Authority and Show Tree View** from the dropdown list as illustrated below:



The **User Authority Manager** panel is displayed in the SOLA workspace. The project name you selected will always be displayed in parentheses as highlighted in **red** beside the panel title:

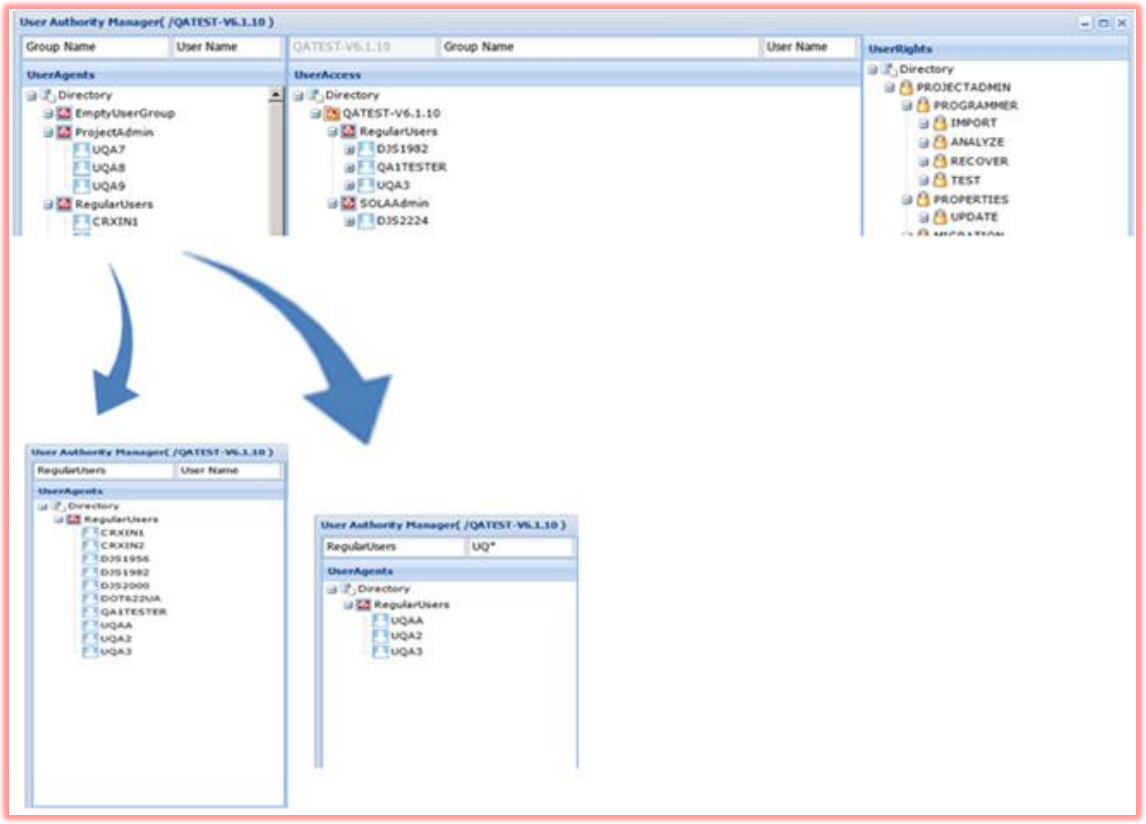


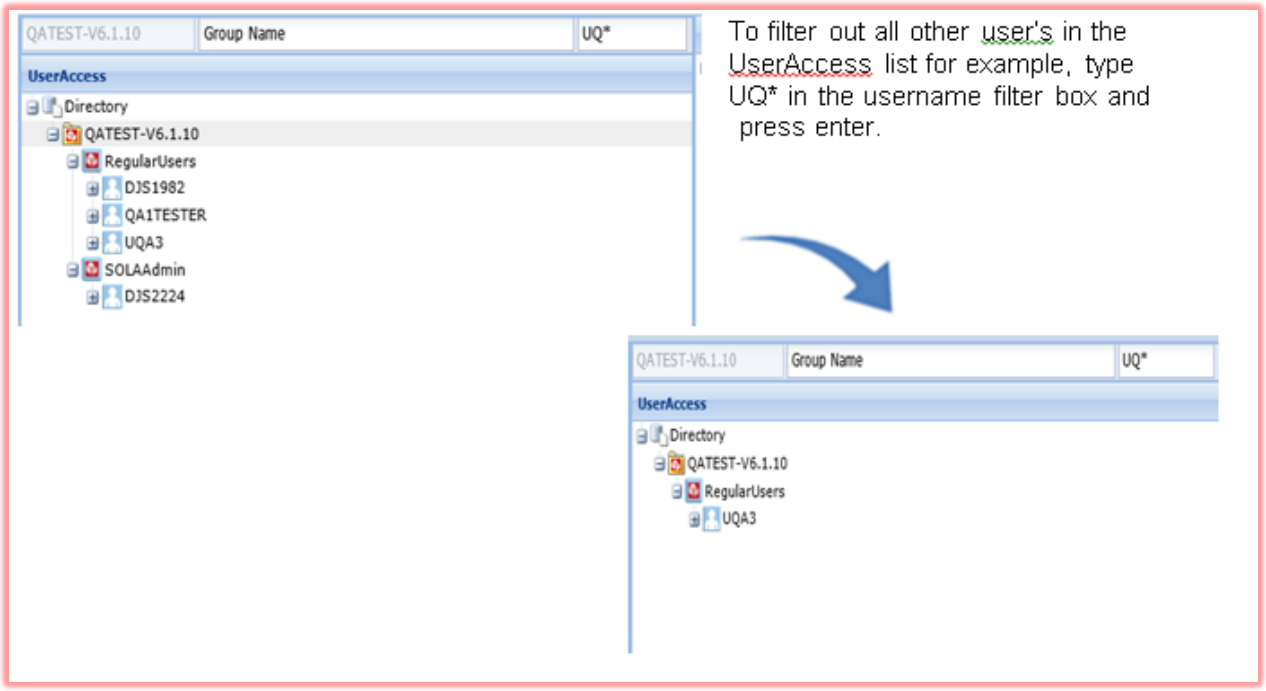
The panel is further broken down with a filter bar above the first and second of three panes described as follows:

- **UserAgents:** the left pane contains all Users defined within their specific groups. Users can be dragged and dropped into a **UserAccess** group in the middle pane for the selected project.
- **UserAccess:** listed in the middle pane are Projects containing the type of access assigned to the User within each User Group. A user can be assigned to different Access/Operation types on different Projects. **UserAgents** ( ) are dragged and dropped into the selected Project ( ), while **UserRights**( ) are dragged and dropped into the Users ( ) in the **UserAccess** pane.
- **UserRights:** the right pane lists Access/Operation types that define what type of access a User is assigned to work on in the project. The operation type is assigned by the SOLA Admin or Project Admin, depending on what authorization level is required.

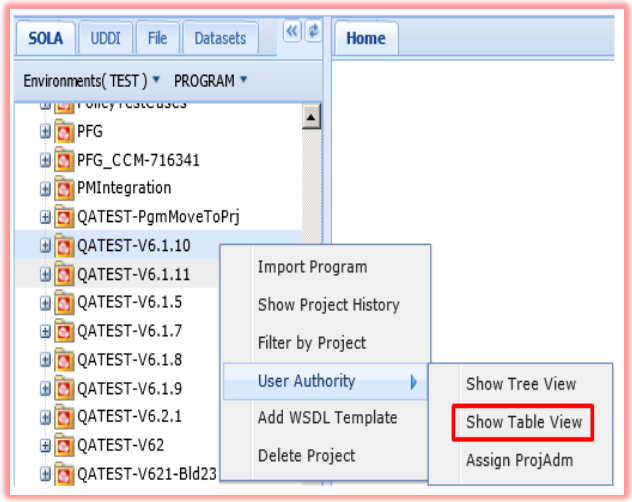
  UserRights( ) are dragged and dropped into the selected Users ( ) in the **UserAccess** pane. **UserRights** are organized in hierarchical order and users can be either assigned individual leaf level authorities like 'ANALYZE', 'TEST', 'PROMOTE' or parent level authorities like 'PROGRAMMER', 'MIGRATION', 'PROJECTADMIN' that inherits all the underlying authorities grouped under that parent level authority.

The Filter bar located above the **UserAgents** and **UserAccess** panes is used to filter the Directory contents of each pane by Group and/or User Name. An example of this can be seen by typing the **RegularUsers** Group Name in the filter box; only the users in that group will be listed. Further filtering of the users beginning with UQ can be done by typing **UQ*** in the User Name filter box.





Show Table View: Another way to view User Access is to display and/or remove user access using the **Show Table View** option in the **User Authority** drop down for a project. More information can be found on this topic by clicking the following link that will take you to Page 232.



Assign ProjAdm: Used by the SOLA Admin to create the Project Admin when there are no Users defined to the project.

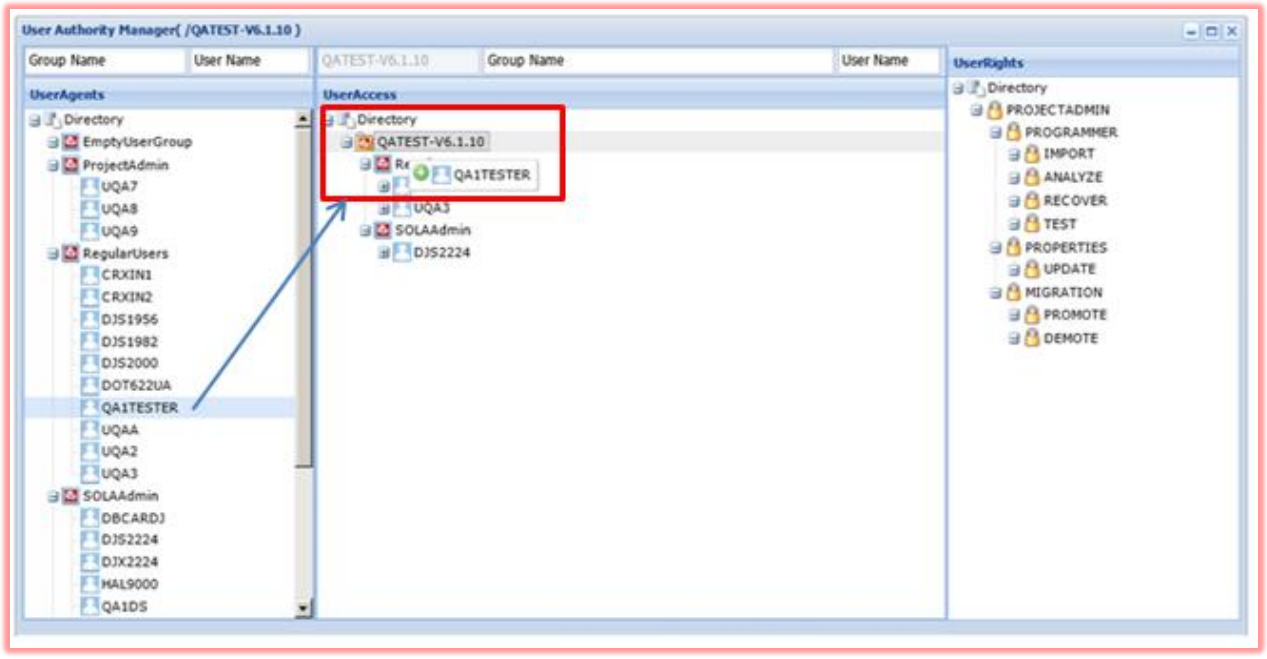
Note: If you want to delete a User who is assigned as a Project Administrator you must first assign another User as a Project Administrator for the project, and only then can you delete the User from the project.

Adding Users to a Project

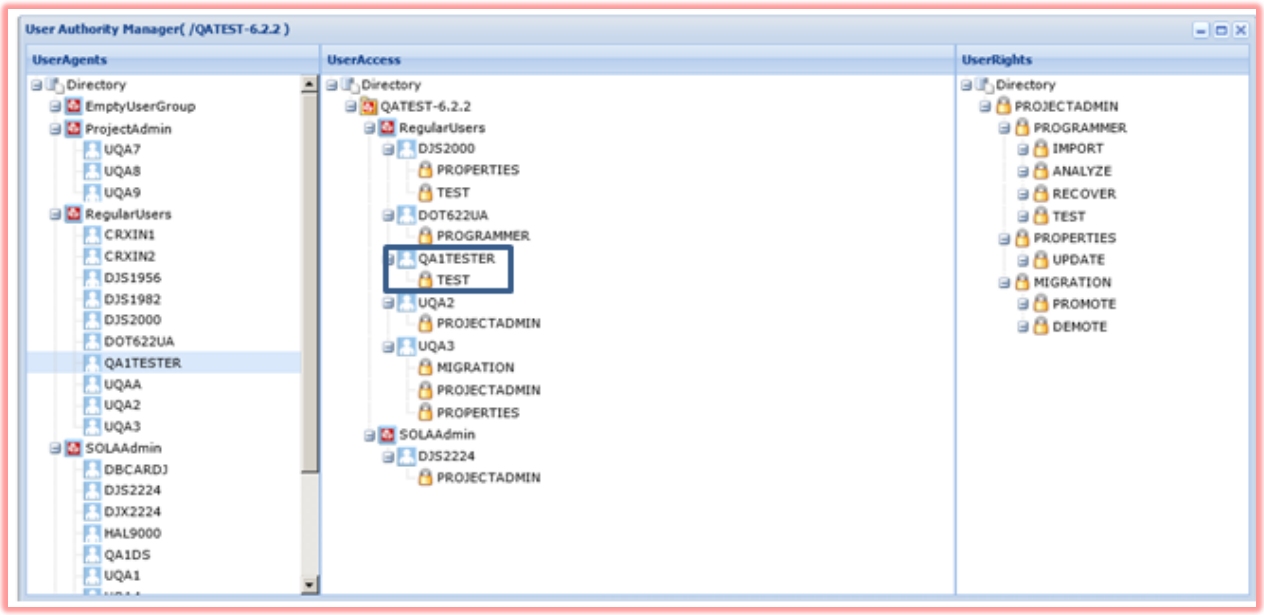
Follow the steps below to add a User to a Project:

1. Select the project from the Directory tree that you want to add the User into.
2. Right click on the project and from the drop down select **UserAuthority** and **Show User Tree** from the Directory tree drop down.
3. The User Authority Manager panel will be presented in the workspace.
4. Locate the User in the list of UserAgents (left pane).
5. Drag and drop the User from the UserAgents list onto the UserAccess **project** (middle pane) you want the User to have access to. Dragging the mouse to the project, you will see a green icon ( ) beside the UserAgent icon indicating a valid move operation.
6. The User will be created and placed in the tree in the UserAccess group within the project you dropped the User onto.

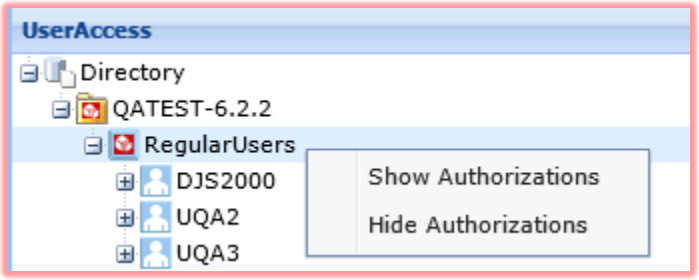
The User QA1TESTER has been dragged and dropped from the left to the right pane onto Project QATEST-6.1.10 in the UserAccess pane. Note the **green** icon beside the **blue** UserAgent icon and username indicating this is a valid move of the user into the project.



The **UserAccess** list of Groups and Users is expanded out to list all of the **UserRights** authorized for each User. All new users are assigned default user rights of **TEST**. As seen below **QA1TESTER** will only be able to use QuickTest to test finalized Methods, until other user rights are applied enabling the user to perform additional SOLA functions.



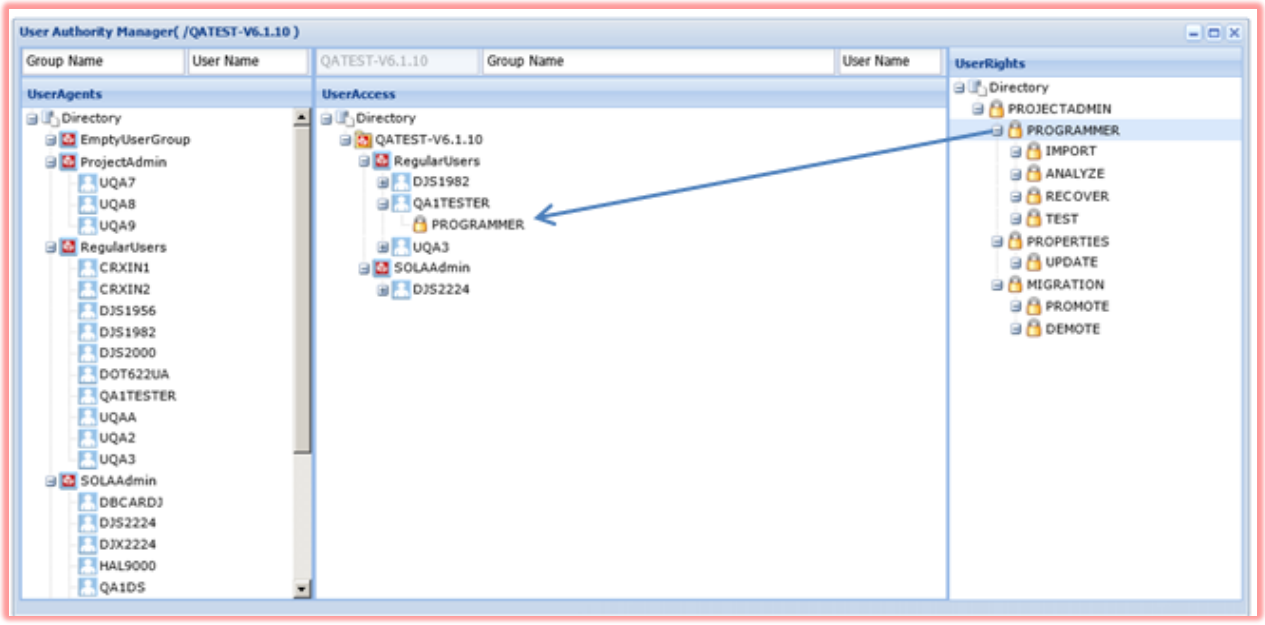
Right click on a Group, Project or 'Directory' nodes in **UserAccess** panel for options to 'Show' or 'Hide' user authorizations.



Assigning UserRights for UserAccess

Once a user has been added to the Project then UserRights can be assigned to that user by dragging and dropping it onto the User in the UserAccess pane (you will see 

). In the illustration below **PROGRAMMER** access was assigned to user **QA1TESTER** overriding the default **TEST** automatically assigned when the user was created:



ProjectAdmin: grants full access to the project, which includes the ability to delete the project, add or remove users and all other actions including promote/demote.

Programmer: grants User access to the project to any or all of the following three: Import, Analyze, Recover and QuickTest.

Import: allows the User to import programs.
Analyze: allows the User to create methods.
Recover: allows the user to recover a method from the method history screen
QuickTest: allows the User to test methods using the SOLA test harness

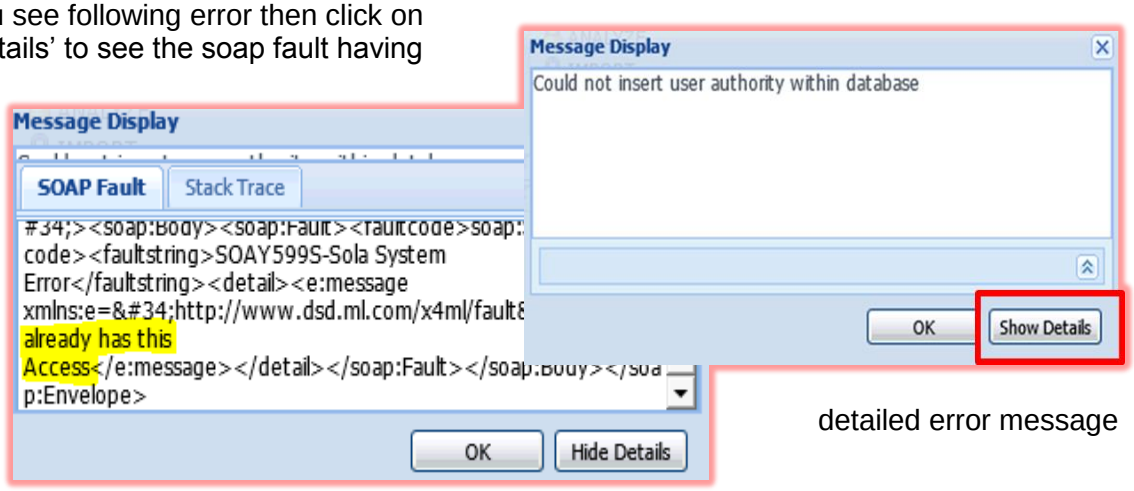
Properties:
Update: allows the User to update properties for the project and its programs and methods

Migration:
Promote: allows the User to promote (move to a higher ranked environment, e.g. promote from STAGE to PROD) programs in the project.
Demote: allows the User to demote (move to a lower ranked environment, e.g. demote from STAGE to TEST) programs in the project.

Note: The SOLA Administrator will need to grant RACF access to the User for any legacy/mainframe datasets used during the importing of a program.

Note: Assigning **PROGRAMMER** authority provides the user with all four lower level accesses, **IMPORT**, **ANALYZE**, **RECOVER** and **TEST**. If you try to drag and drop one of the four lower level accesses onto **QA1TESTER** you will get an error 'Could not insert user authority into Database'. This is because the user already has access as **PROGRAMMER.** All higher levels of access i.e. **PROJECTADMIN** has authority to every access beneath it, while a user assigned only with **MIGRATION** can only access the lower level functions beneath it.

When you see following error then click on 'Show Details' to see the soap fault having



detailed error message

The SOLA Administrator can grant access to Policy Admin or Tester Admin SOAP Test (RAW TESTER) and restricted endpoints, to any Project Admin or Programmer via the Resource Manager Users/Policies tab (*see the Resource Manager Users Guide for further information on administering PolicyAdmin & TesterAdmin authority*).

The following access matrix illustrates the access to QuickTest and Raw Soap-Test screens and types of end-points that user has access to depending on user role and access to Tester Admin role. (*see the Resource Manager Users Guide for further information on administering Testeradmin authority*).

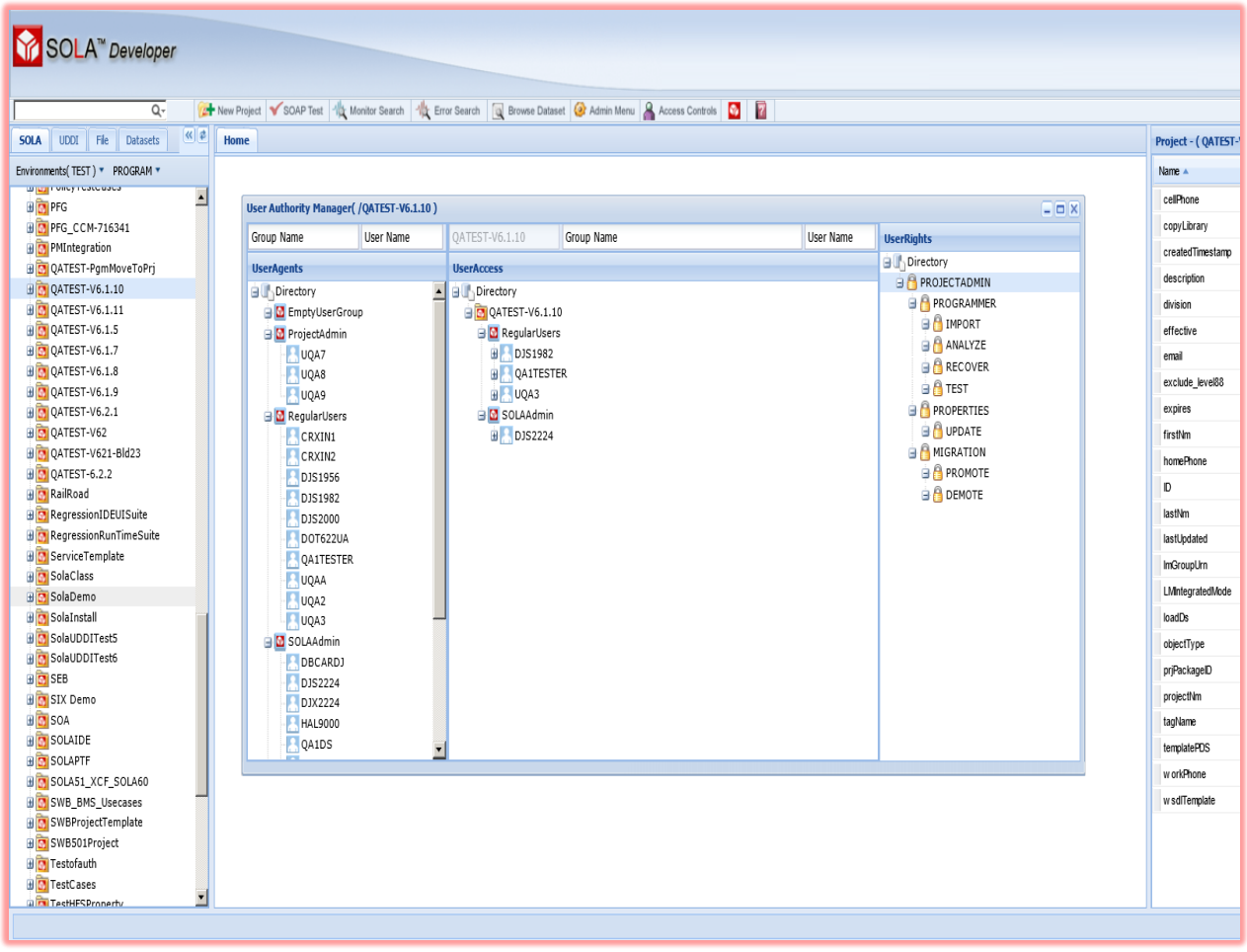
Project Admin	Tester Admin	Quick Test		Raw Soap-Test	
		Open End-points	Restricted End-points	Open End-points	Restricted End-points
Y	Y	Y	Y	Y	Y
Y	N	Y	N	N	N
N	Y	Y	N	Y	Y
N	N	Y	N	N	N

Cloning UserAccess

Cloning a user access allows you to quickly copy access rights of one SOLA user to another. Cloning of user access can be done on users either within a project or across two projects. **UserRights** can be cloned between user from one project to user in another.

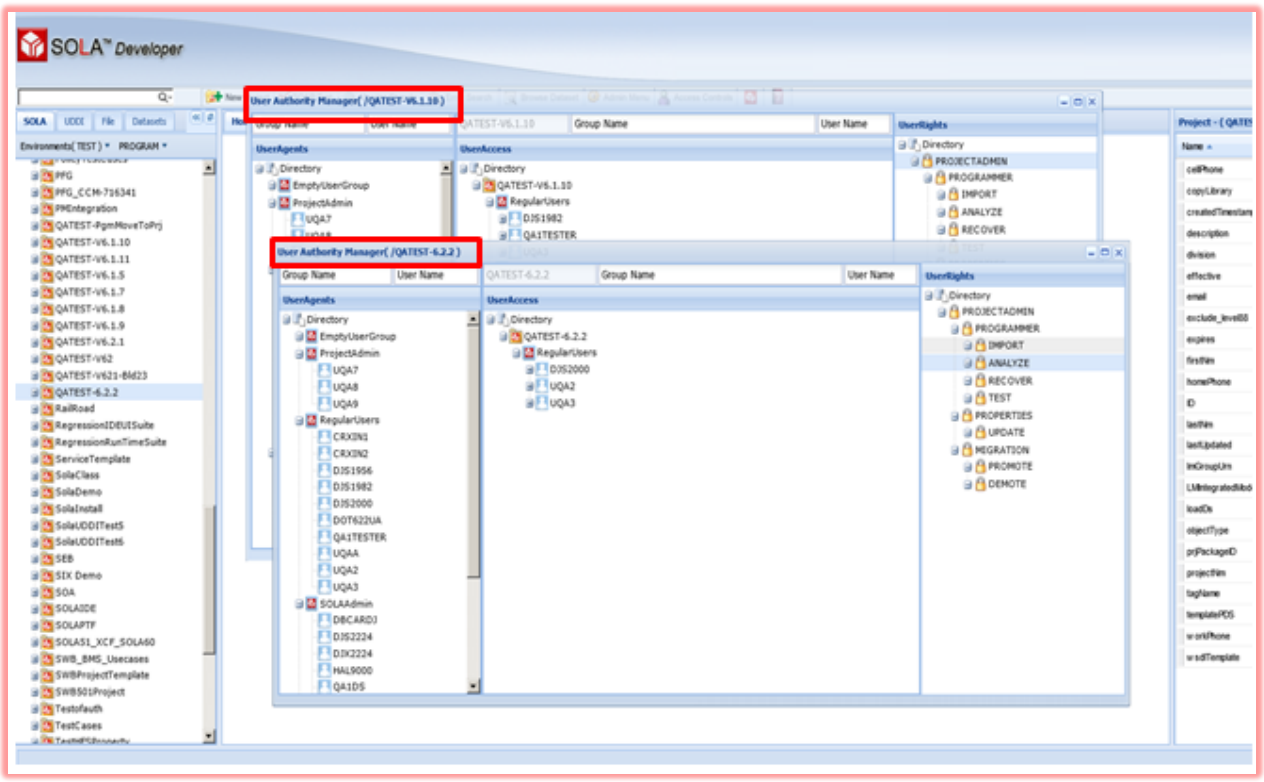
To clone the **UserAccess** for a User into another project you will need to first locate the project containing the User you want to clone, right click on **User Authority and Show User Tree**. The User Authority Manager panel (Figure 1) will appear in the workspace for the project you have selected.

Figure 1:



Next, locate the project in the SOLA Directory tree containing the User you want to become cloned; right click **User Authority and Show User Tree**. The next User Authority Manager panel (Figure 2) will be presented in the workspace underneath the first panel containing the user to be cloned.

Figure 2:

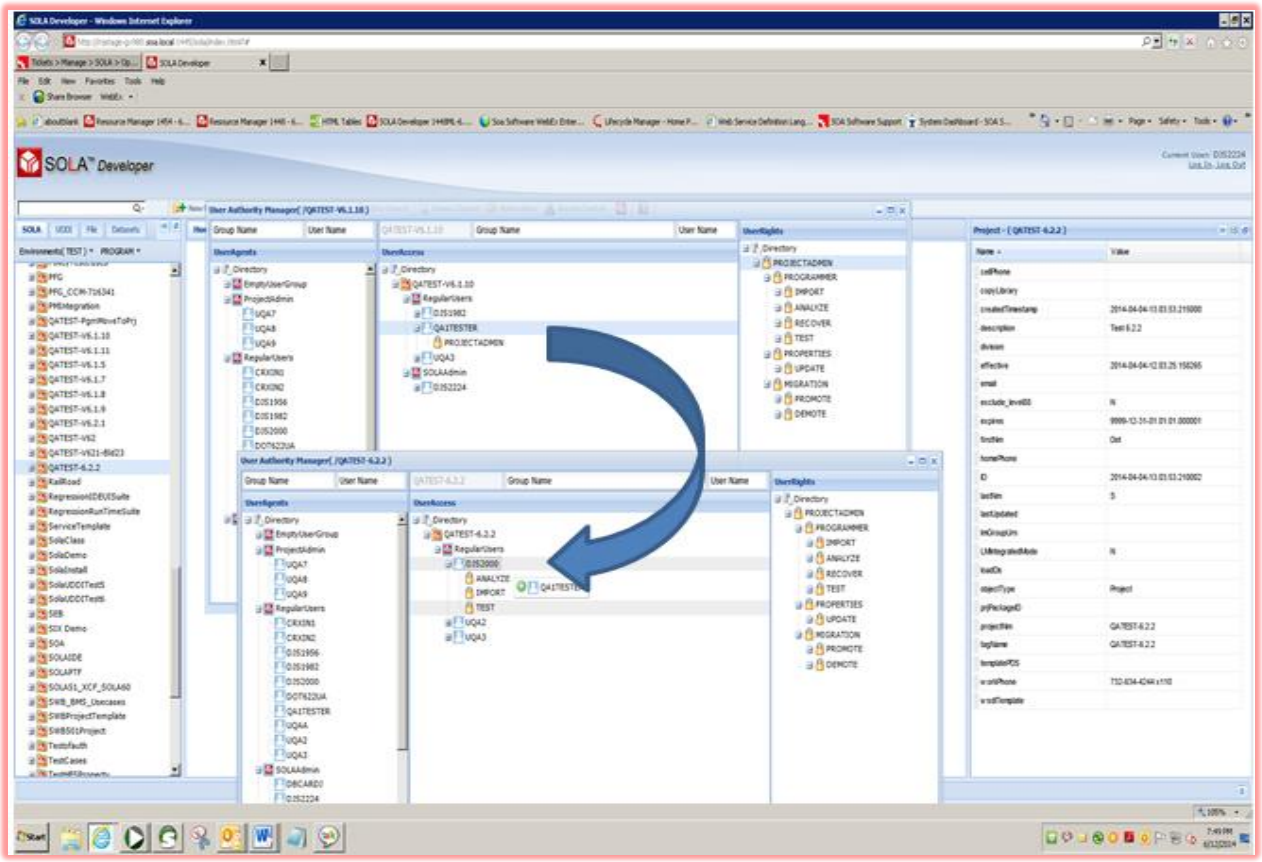


Note: You will need to click on each panel to drag and drop it into the proper position in the workspace which will enable you to locate the user's you will be working with.

In Figure 1 above, User **QA1TESTER** assigned to project **QATEST-V6.1.10** contains UserRights we will be cloning for User **DJS2000** working in project **QATEST-V6.2.2** in Figure 2. After each panel is positioned and each user is visible you can drag and drop **QA1TESTER** onto **DJS2000** as seen in Figure 3 below:

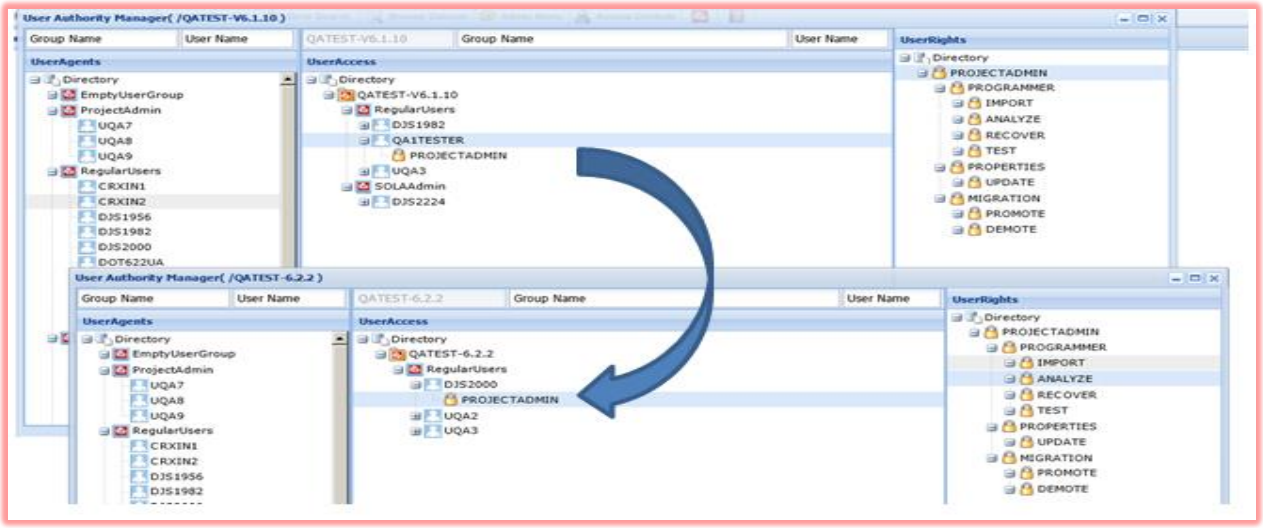
User **QA1TESTER** has **PROJECTADMIN** access and after the cloning operation is completed user **DJS2000** will also have the same access. Below in Figure 3 you will see the UserRights that user **DJS2000** had before the cloning began.

Figure 3:



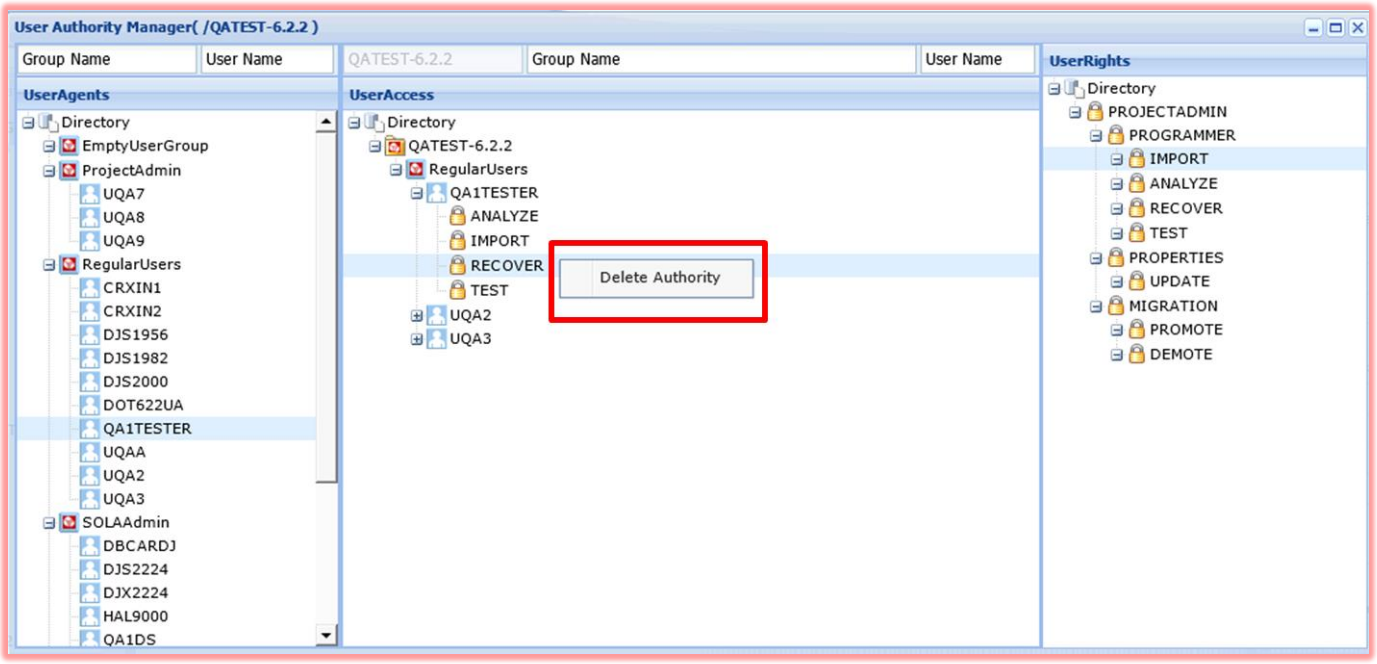
User **DJS2000** now has **PROJECTADMIN** UserRights as seen below in Figure 4:

Figure 4:



Delete User Authority

To delete user authority right click on the authority of the user to be deleted and select 'Delete Authority' option as shown below

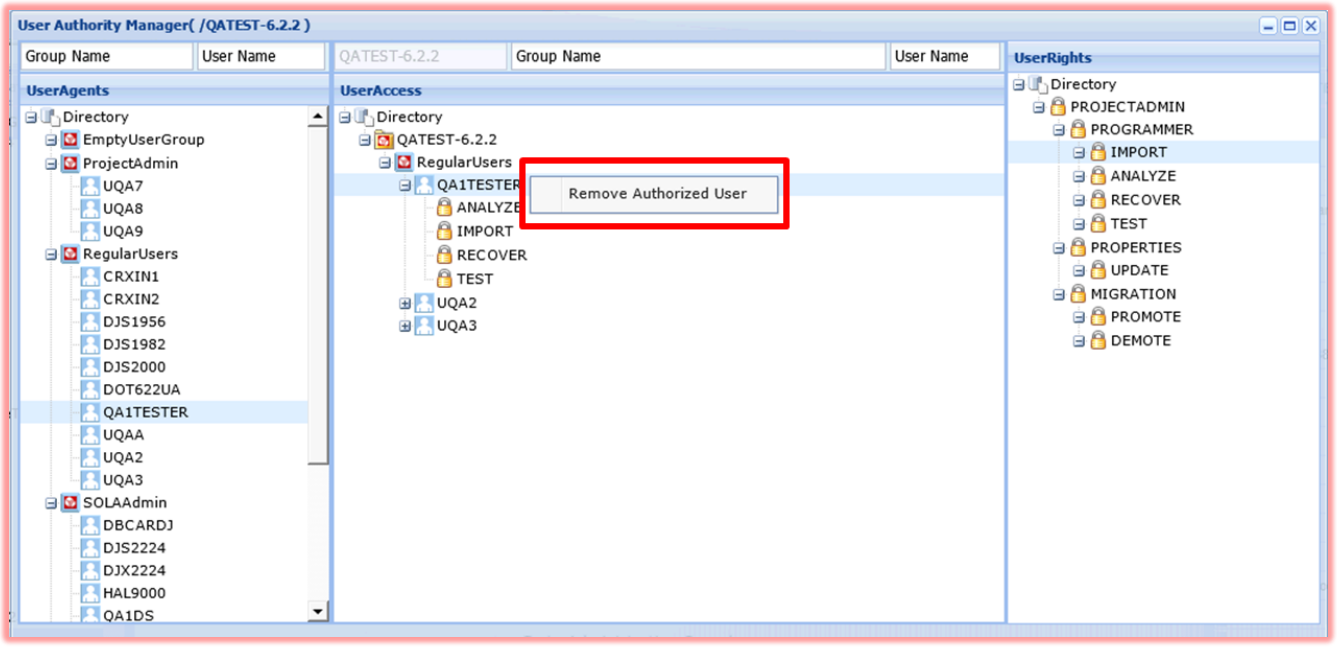


Deleting the last authority of user will restore 'TEST' authority as no users can be assigned to a project without a role.

If a user with 'PROJECTADMIN' authority is being removed then SOLA checks to ensure there are alternate user(s) of project who have been assigned 'PROJECTADMIN' role. If there is no other user assigned as project administrator then the 'Delete Authority' function will fail and will indicate that the user is the only the project administrator and so the authority cannot be removed.

Remove Users from project

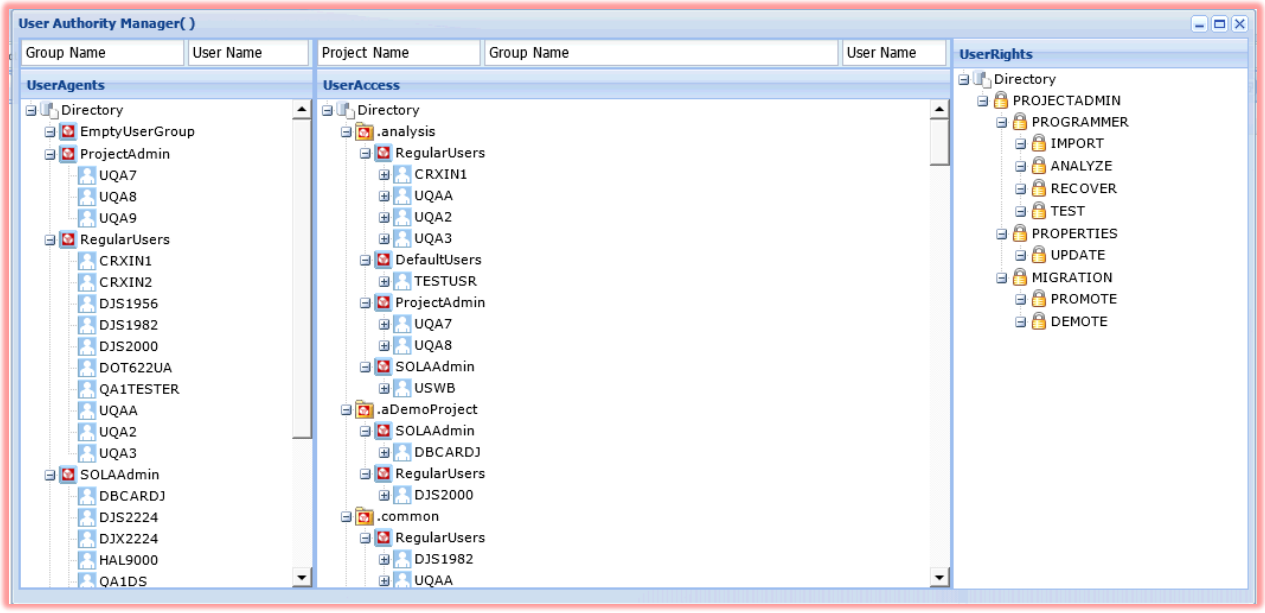
To remove authorized users from a project right click on the user to be removed and select 'Remove Authorized User' option as shown below



If a user with 'PROJECTADMIN' authority is being removed then SOLA checks to ensure there are alternate user(s) of project who have been assigned 'PROJECTADMIN' role. If there is no other user assigned as project administrator then the 'Delete Authority' function will fail and will indicate that the user is the only the project administrator and so the authority cannot be removed.

Additional User Authority Access for SOLA Administrators

SOLA Administrators have an additional access to the global user access tree across projects by right clicking on the 'Directory' node on project tree and select 'User Authority' option as shown below



SOLA Administrator has additional capability to filter based on Project Names.

Using SOLA Developer - Commarea

SOLA can create web services in which the mainframe acts as a server (inbound), and as a client (outbound).

- **Inbound** (mainframe as server):
  - **Bottom-up:** start with a Commarea program and create a WSDL, metadata template, test harness and UDDI entry by analyzing the program's interface.
  - **Meet-in-the-Middle:** start with a WSDL **and** a copybook, and create a metadata template, test harness and UDDI entry by merging the WSDL and copybook.
  - **Top-down (WSDL-First):** start with a WSDL and create a COBOL or PL/I copybook.

- **Outbound** (mainframe as client):
  - **Top-down (WSDL-First):** start with a WSDL and create a COBOL or PL/I copybook that will be used as the interface between SOLA and an outbound web service.

Note: SOLA uses the terms '*Class'* to refer to a web service and '*Method'* to refer to a web service operation.

Creating an Inbound Web Service from a Commarea Program – Bottom Up

This section will describe the steps necessary to create a web service from a COBOL or PL/I commarea program using "bottom up" methodology. Bottom up means that you will be starting with either a compile listing or a copybook and using SOLA Developer to import the program and create methods from the program's various functions. The end result will be a WSDL, metadata template, test harness and a UDDI entry.

Creating a web service from a commarea program is a two-step process:

1. Import the program and create a *Class*
2. Analyze the *Class* to create *Methods*

The Import procedure is a single step operation that consumes the program (or copybook) and documents it in the SOLA Directory as a *Class*. No other artifacts are produced.

The Analysis procedure takes a *Class* and creates a *Method* (web service operation). It also creates four artifacts:

1. Run Time metadata (called a *Template*)
2. Test Harness
3. WSDL
4. SOLA Directory entries for the method

You can import a commarea program from the following sources:

- **Compile Listing:** the preferred import method. Importing from a saved compile listing allows SOLA to determine information about the program being imported, such as field types (input, output, etc.), usage and more.
- **Job Name and Number:** if the compile listing is in the JES output queue, you can import the program using the job name and number. This gives the same benefits as importing a saved compile listing. In order to import from a job name and number, your sysout files must be routed to your installation-defined held output queue.
- **Copybook:** although programs can be imported from copy books, SOLA will not be able to automatically configure the program as it can with the other two methods.
- **Multiple Datasets:** you can also import from more than one copybook (all copybooks are concatenated into a single WSDL).

Step 1 – Mainframe Preparations

Depending on your CICS installation, you may need to create a PPT entry for the Run Time metadata and a PPT entry to dispatch DPL requests for your program from the SOLA Web Owning region to your Application Owning region.

Compiler Options

The imported program will need to be compiled with the **MAP** compiler option. Here is an example of some Compiler options that can be used to compile a program for use with SOLA:

```
 EDIT       DBSOLA.SOLA.JCL(COMPBAT5) - 01.01              Columns 00001 00072
 Command ===> sub                                            Scroll ===> CSR
 ****** ***************************** Top of Data ******************************
000001 //DBSOLAA JOB (A),'N',CLASS=F,MSGCLASS=Y,NOTIFY=DBSOLA
000002 //COB22   EXEC PGM=IGYCRCTL,
000003 // PARM=(,
000004 // 'ARITH(EXTEND),OBJECT,RENT,APOST,TRUNC(OPT)',
000005 // ',OPT(FULL),NUMPROC(PFD),XREF(FULL),MAP,LIST',
000006 // )
000007 //SYSIN    DD  DSN=DBSOLA.SOLA.COBOLO(CONVERT),DISP=SHR
000008 //STEPLIB  DD  DISP=SHR,
000009 //             DSN=SYS1.SIGYCOMP
000010 //SYSLIB   DD  DSN=DBSOLA.SOLA.COBCOPY#,DISP=SHR
000011 //SYSTERM  DD  SYSOUT=*
 IKJ56250I JOB DBSOLAA(JOB16902) SUBMITTED
***
```

The SOLA Import process only needs the Compile listing, there is no need to link-edit the program to create a new load module.

Once the program has been compiled, it can be imported either directly from the JES output queue or from a dataset that the compiler output is saved in.

Alternatively, you can Import a COBOL copybook. This method, although effective, doesn't allow SOLA to determine the inputs and outputs for the program.

Note: If you Import a compile listing and you use Intertest for debugging, then you shouldn't use the Intertest CUTPRINT option because this option can eliminate parts of the compile listing that are used by SOLA's Import process.

Environment Setup

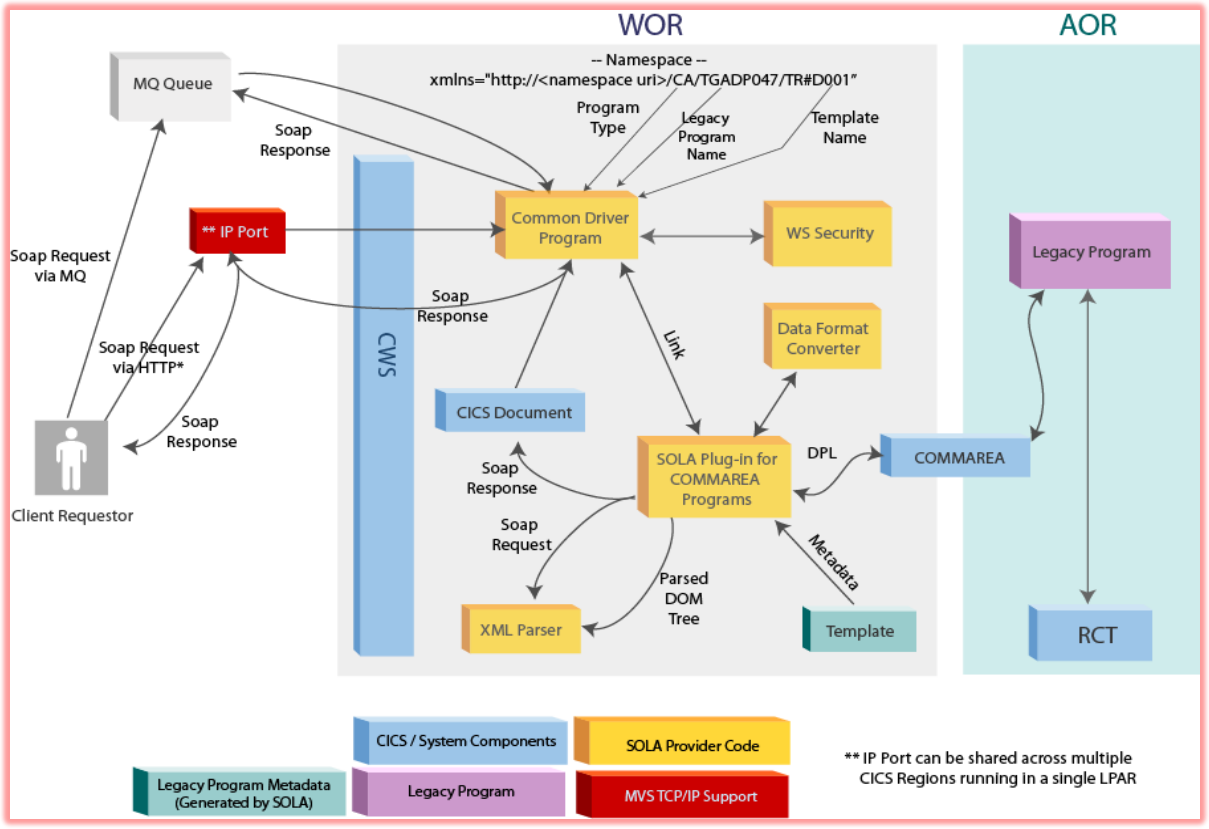
Before you are able to use a web service created with SOLA, you will need to perform some setup operations for the SOLA Run-time.

To understand why this is necessary, it may help to understand the SOLA Run-Time architecture. SOLA is built using MRO (Multiple Region Operation). In an MRO environment, there are a minimum of two CICS regions that are involved in performing work – a Terminal Owning Region (TOR) and an Application Owning Region (AOR). Because SOLA uses the CICS Web Support features we refer to the TOR as a WOR (Web Owning Region).

TECH TIP

MRO (or Multiple Region Operation) is the term used to describe a set of inter-linked CICS regions. Each region usually performs a different function, and requests pertaining to those functions are routed to the appropriate region.

In MRO, the Endpoint SOAP URL points to the WOR. The WOR accepts work and forwards it to the appropriate AOR. Commarea programs are run in an AOR. The following is a diagram of the architecture for commarea programs.



SOA Enabling Commarea Programs with SOLA

The conversion of SOAP messages into and out of a commarea is done in the WOR. To perform the conversion, SOLA references the template in the WOR and links to the legacy program in the AOR (this is known as a DPL, or Distributed Program Link).

Before you can execute the new web service, you will need to set up some CICS table entries. In the WOR you need PPT entries, one for the template and one for the legacy program, specifying it as remote. In the AOR you need one PCT entry to accept the link from the WOR.

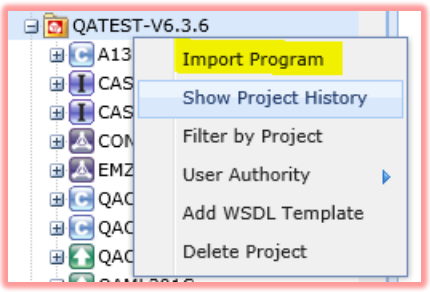
The table below lists sample entries based on a program named CONVERT and its template, named CONVD001. The program CONVERT, which you will be using for the examples in this

section, is a sample program that is shipped with SOLA and can be found in the SAMPLIB library.

WOR	**AOR**
DEFINE PROGRAM(CONVD001) GROUP(SOLAGRP) LANG(ASSEMBLER) STATUS(ENABLED)	DEFINE TRANSACTION(CON#) GROUP(SOLAGRP) PROGRAM(DFHMIRS)
DEFINE PROGRAM(CONVERT) GROUP(SOLAGRP) LANG(LE) REMOTESYSTEM(aorx) TRANID(CON#)	

The final required setup step is to issue a new copy command in the WOR region for the legacy program's template.

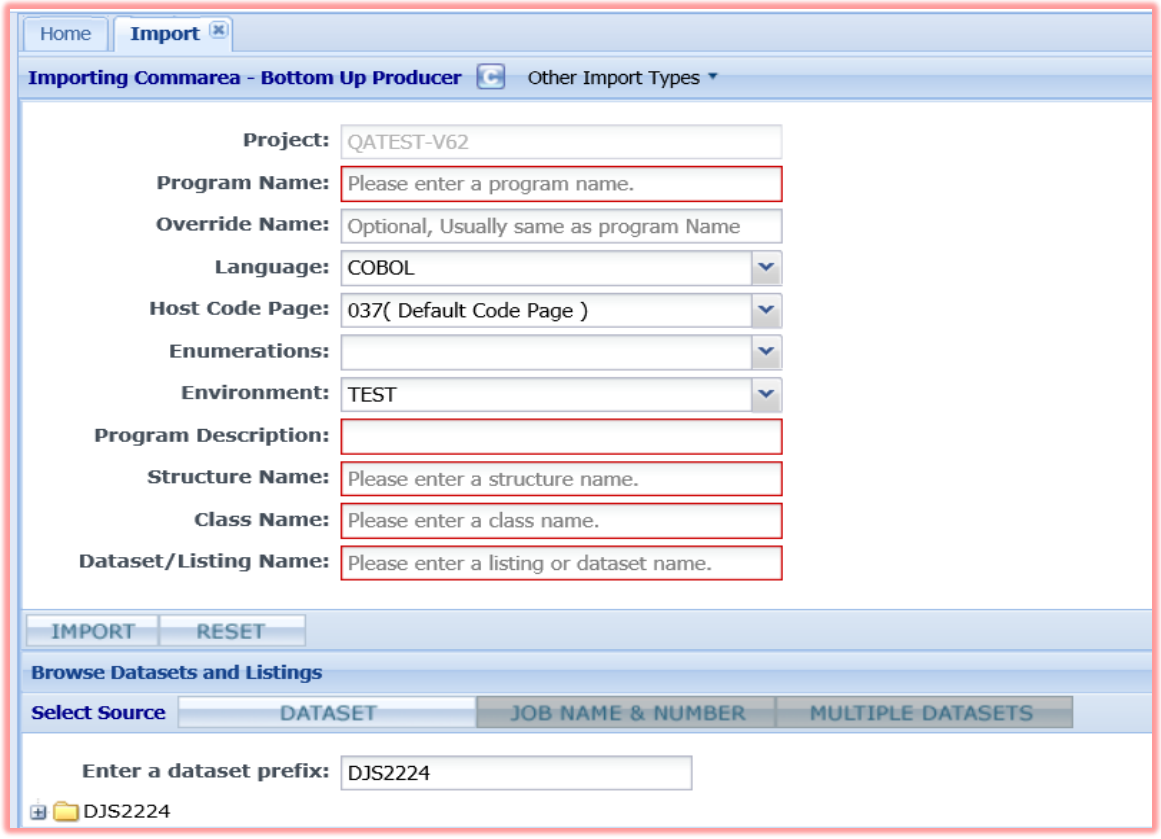
Step 2 – Importing a Commarea Program



Select the project you wish to import to and right-click it. From the pop-up menu, select **Import Program**. If you wish to import the program to a new project, first follow the steps for creating a new project on page 22.

In order to import a program into a project, you must be an authorized user of that project. Once the program is imported, you can drag and drop the program from one project to another. However, you must also be an authorized user of the project you wish to move the program into.

After you select **Import Program**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.



The default program type is commarea bottom-up, though you can change that by using the **Other Import Types** menu.

The Import panel consists of a series of fields used to provide information about the source program and the destination SOLA program that will be created.

Fields outlined in **red** are required. The **red** outline disappears when the field is populated.

- **Project:** this field is pre-populated and contains the name of the project into which the program is being imported. Although it cannot be changed during import, you can drag the program into a different project after it has been imported.

- **Program Name:** the name of the SOLA program that you will create. This name does not have to match the name of the source program, but if it does not, then the Override Name must do so. The program name is limited to eight characters, whether it matches the target program name or not.

  Program Name can be used for program versioning if you need multiple versions of the same program in use at the same time (as opposed to existing only in history). Assigning alternate (in this case version based) names in this field will create programs that occupy distinct namespaces within the WSDL generated by SOLA. For example, if we import the same program and use two different program names, myProgV1 and myProgV2, they will have different namespaces:

  http://searchByName.ClientFinder.x4ml.soa.com/CA/SOLACA04/MYPROGV1
  http://searchByName.ClientFinder.x4ml.soa.com/CA/SOLACA04/MYPROGV2

  Both versions will call the same legacy program, but because SOLA treats them as distinct programs there are no restrictions regarding method names or method functionality (e.g. the programs can have the same exact methods, methods with the same names but different functionality, methods with different names, or any combination thereof).

  When using Program Name for versioning, the Override Name field must contain the actual eight character name of the target legacy program.

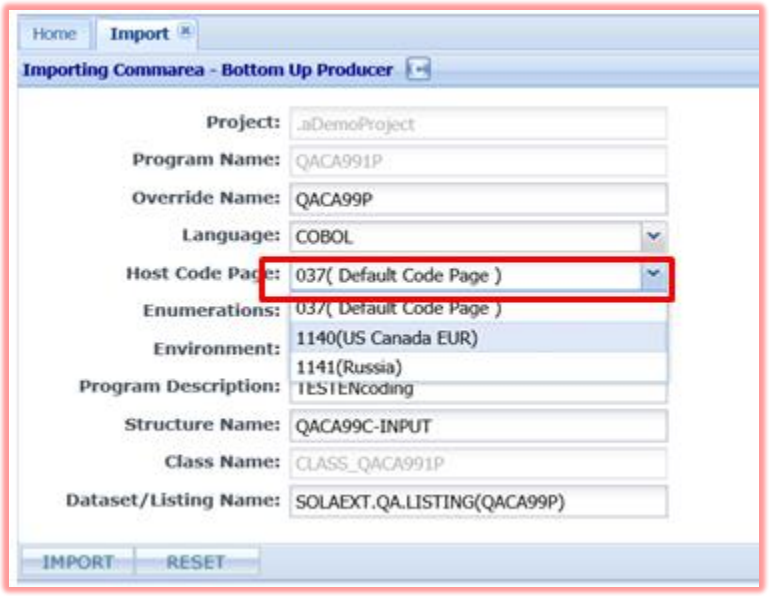
- **Override Name:** The name of a target program to execute. Use this field when the target name differs from the program name (for example, when using the Program Name field for versioning).

- **Language:** the language the source program is written in. Choices are COBOL, PL/I or Natural.

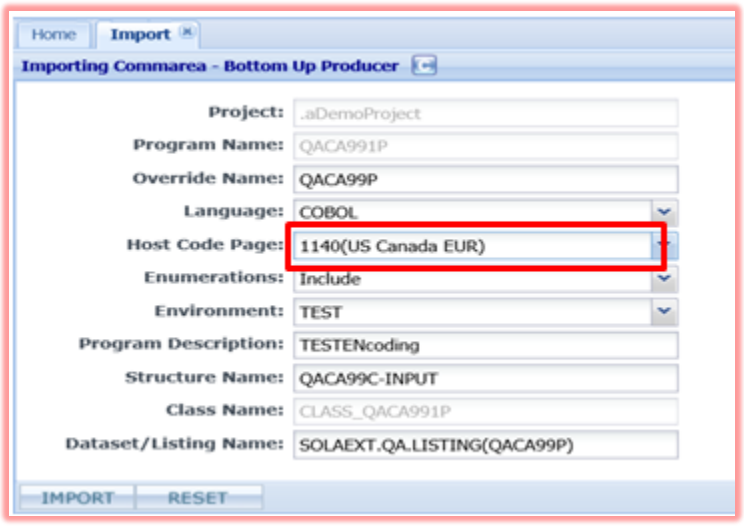
- **Host Code Page:** values entered by the Administrator into the Admin Menu / Property File – **codepage.xml** that enable user to choose code page conversion to and from UTF8 are displayed here. The default is EBCDIC code page 37; CCSID 1140 is supported and is the Euro currency update of code page CCSID 37. In that code page, the "¤" (currency sign) character at code point 9F is replaced with the "€" (Euro sign) character.

  Note: Code page values stored in codepage.xml and selected during IMPORT are validated when the service is executed. It is important to make sure the code page you are using during IMPORT is valid.

Changing FROM: Default Code Page:



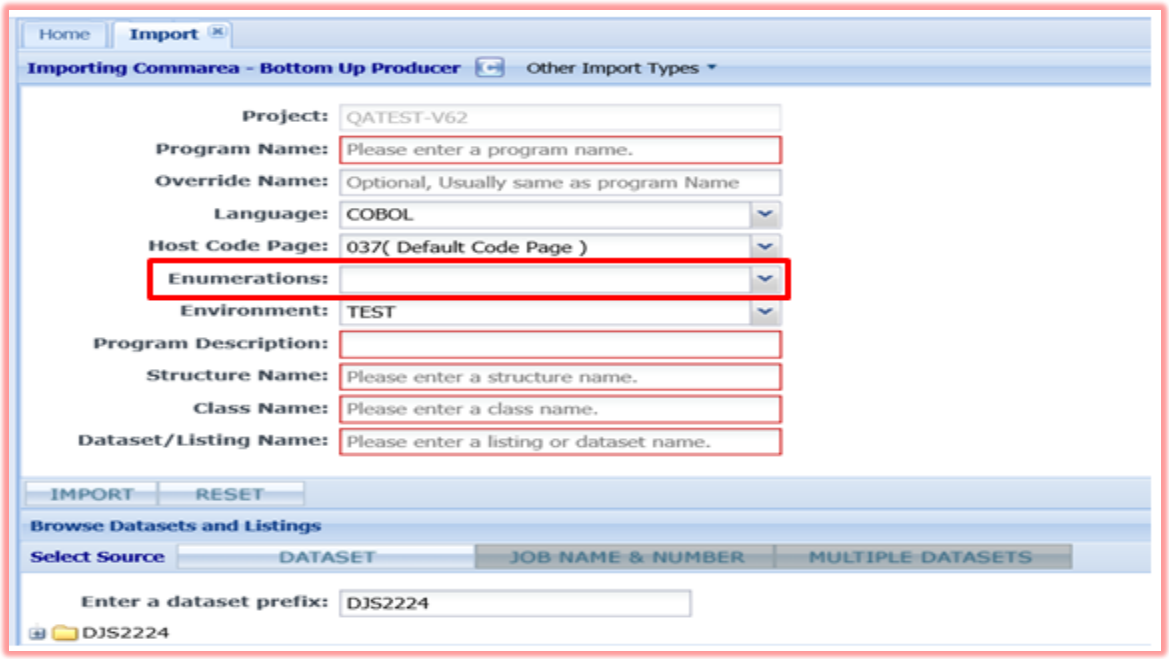
TO: Host Code Page 1140(US Canada EUR):



When selecting a Host Code Page other than the default code page 37, the soapAction generated in the WSDL will contain the Client Code Page (CCP) and Host Code Page (HCP). This can be seen in the example below:

```
- <binding name="CLASS_QACA991PBindingName" type="tns:CLASS_QACA991PPortTypeName">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  - <operation name="QACA991M">
      <soap:operation style="document" soapAction="/CA/QACA99P/QACA991T/CCP:UTF-8/HCP:1140"/>
    - <input>
        <soap:body use="literal"/>
      </input>
```

- **Enumerations:** If enumerations (viz. 88 level items in COBOL) needs to be imported or not. Choices are Include or Exclude



- **Environment:** the created program's environment. The environment is a custom property in SOLA and available environments will depend on your particular installation. Some examples of environments are "Test", "QA" and "Production".

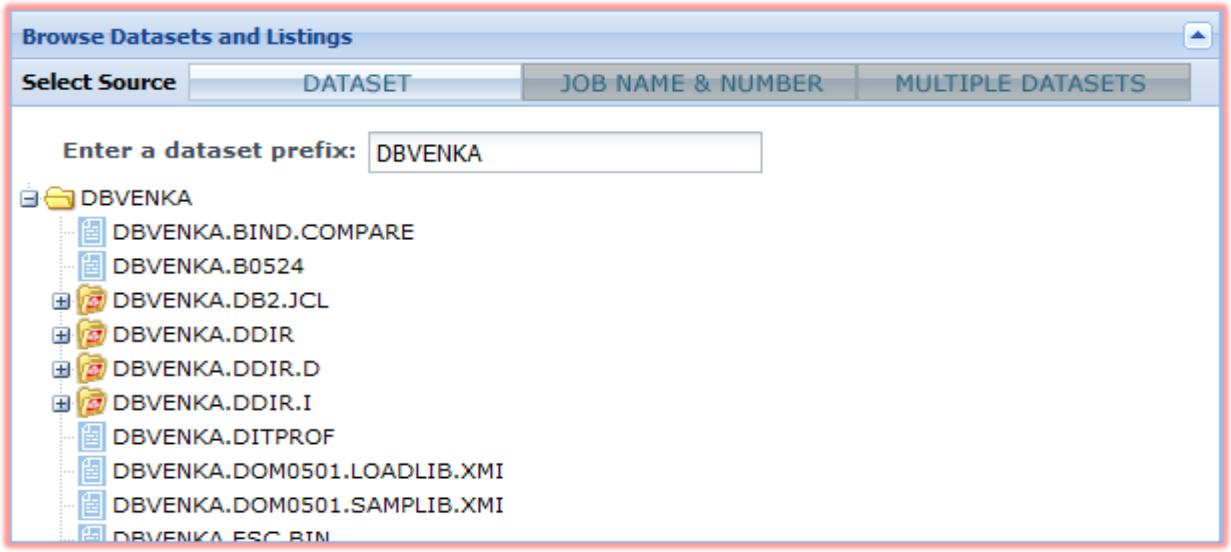
- **Program Description:** a brief free-form description of the program.

- **Structure Name:** the 01 level COBOL structure that describes the interface that your program exposes. This is typically named "DFHCOMMAREA", though the name may vary. If you are unsure of what the structure is called in the program you are importing, you can use the Browse Dataset feature described on page 235, or look at the program in TSO.

- **Class Name:** when you expose a program as a web service, its operations will be exposed as methods. Distributed systems classify methods as belonging to a class. Therefore, SOLA requires that you assign a class name to the program.

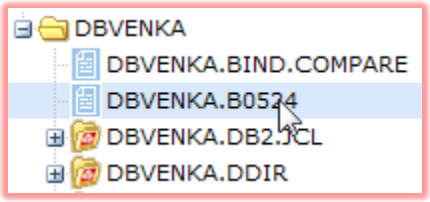
- **Dataset / Listing Name:** the input source. As mentioned previously, SOLA can import a commarea program from a compile listing (either saved or from the JES output queue) or from one or more copybooks. A compile listing is preferred because it allows SOLA to attempt to categorize the interface fields, saving you work during analysis.

At the bottom of the Import panel is the Browse Dataset and Listings panel. This panel allows you to pick the input source from a list without having to manually enter it into the **Dataset/Listing Name** field.



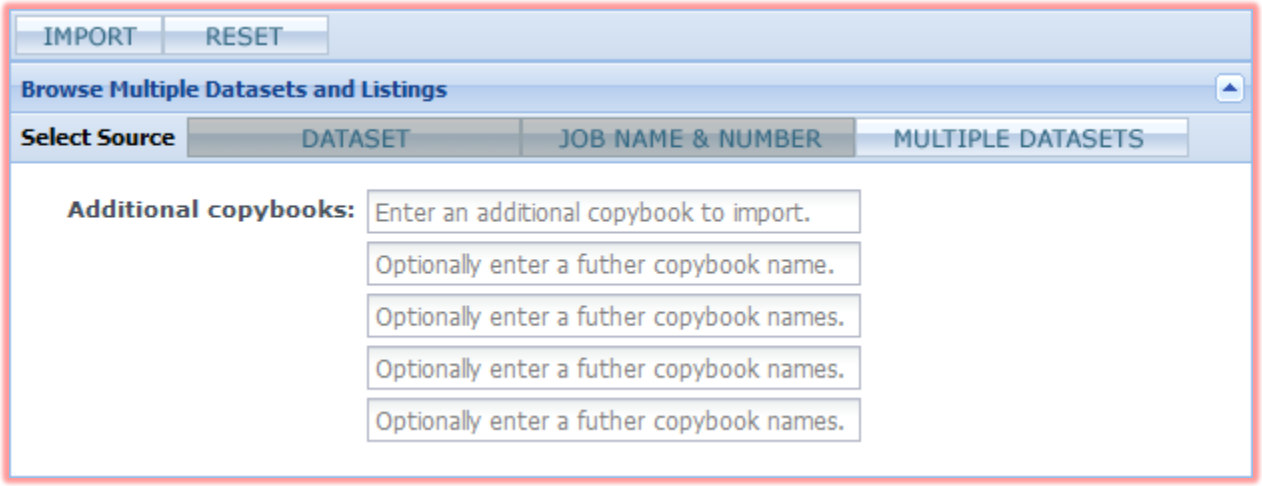
To use this panel, select from one of the three available source types by clicking on the appropriate button tab.

The Dataset option includes both saved compile listings and copybooks. You can change your default dataset prefix by entering a new value in the **Enter a dataset prefix:** field. Your default dataset prefix is a user-level custom property that can be set in your user properties (page 4).



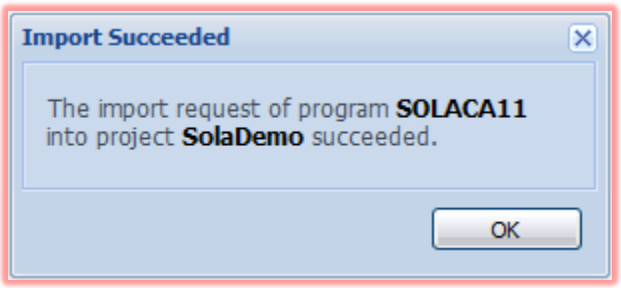
Once you have located the dataset or listing you want to import from, double click the dataset/ listing name to populate the **Dataset/Listing Name** field with your selection.

If you select **Multiple Datasets**, you will not be presented with a directory tree. Instead, you will be given five blank fields that you can use to specify up to five copybooks.



When you have filled in all required fields and are ready to import, click the **IMPORT** button.

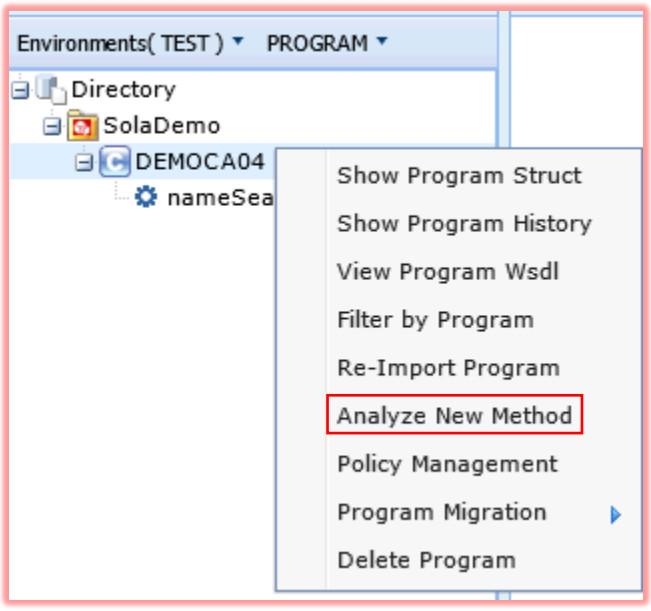
Upon successful import, a confirmation message will be displayed. The newly created program will appear in the SOLA directory under the project you chose to import it into. Once the program is created, you can drag and drop it into any project you wish.



When importing commarea programs, the creation of methods is a separate step from the importing of the program. The following section will detail the creation of a commarea method.

Step 3 – Creating Methods in a Commarea Program

Once a Commarea program has been imported, you can create methods by isolating individual functions within a program. Creating a method is known as *Analysis*. A Commarea program can support a complicated set of requests and responses, while a method is typically a subset of that functionality, sometimes even a single request/response operation. Therefore, a program with complex functionality may require the creation of several methods to expose the full range of its capabilities as web services.

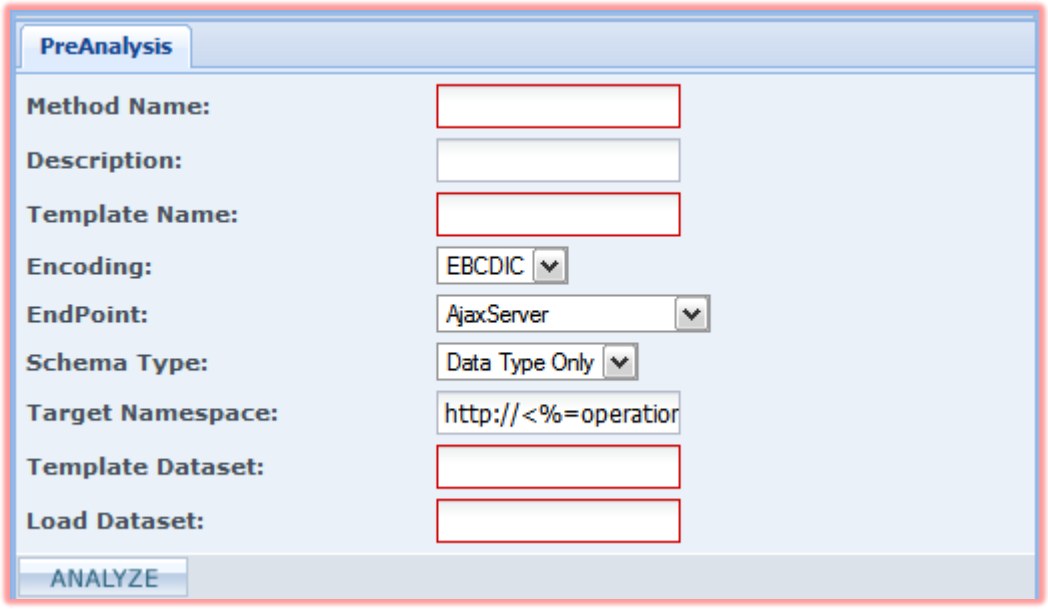


For example, let's take a simple program that converts either temperature (from Fahrenheit to Celsius) or length (from inches to centimeters). When the program is executed, it expects the user to provide a function code for temperature conversion or length conversion.

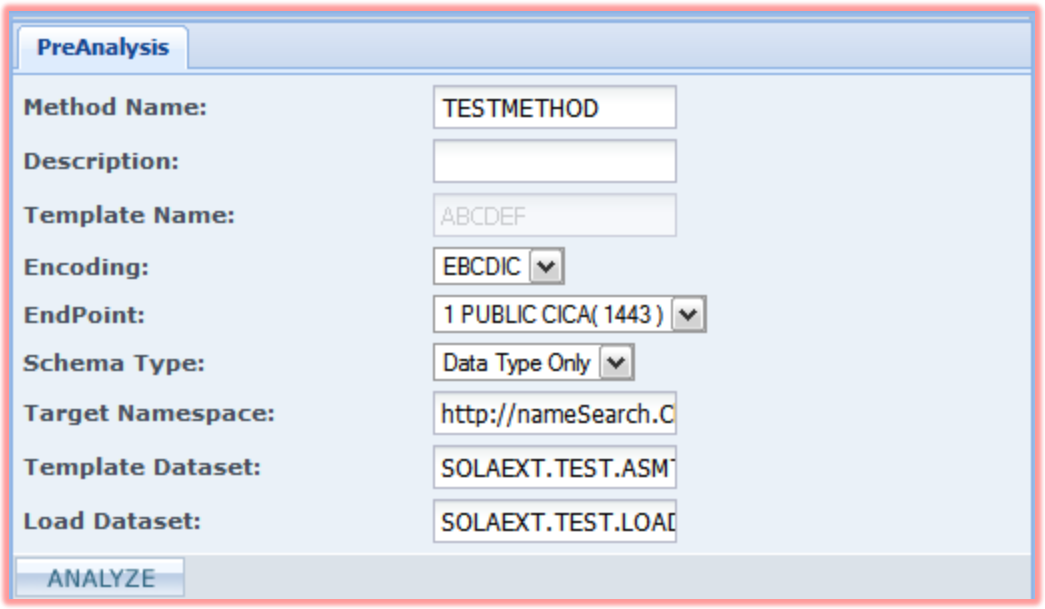
Depending on the function code, the program assumes the input provided is a temperature or a length, and will perform the necessary conversion. When creating methods from this program, you would typically create two separate methods, one to convert Fahrenheit to Celsius and the other to convert inches to centimeters.

When creating methods, keep in mind which environment you are currently working in. Typically, only test environments allow for the creation of methods, though this is configurable.

To create a method, right click the program in the SOLA Directory and select **Analyze New Method**. This will open the SOLA Analysis panel in the workspace.



The Analysis panel consists of a series of fields used to configure the method. Fields outlined in red are required. The red outline disappears when the field is populated.



- **Method Name:** the name of the method being created. The method name will be used in the WSDL as the operation name, and will also appear in UDDI searches.

- **Description:** a brief description of the method.

- **Template Name:** the name of the template (run-time metadata) that will be created for this method. The template name must be unique and must conform to Partitioned Data Set (PDS) member naming conventions. The template tells SOLA how to convert a

SOAP request into a legacy commarea, and how to convert the legacy commarea into a SOAP response. A template will be assembled by SOLA into an Assembler Data Only Load Module.

- **Encoding:** the data format that SOLA will deliver to your program when executing the method. Options are EBCDIC (default) or ASCII. This option is provided for programs that were originally coded to accept ASCII data, and which internally convert the ASCII data to EBCDIC and vice versa.

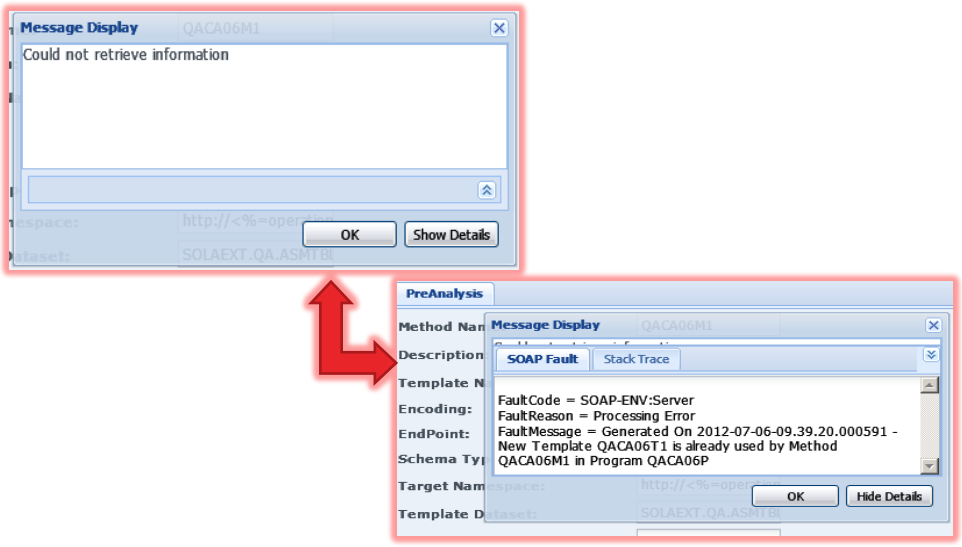
- **End Point:** the location of the SOLA SOAP server. Options will vary based on your installation.

- **Schema Type:** when you analyze a method one of the artifacts that you're creating is WSDL. The WSDL will contain a schema, which is a description of the input and output messages used by this web service. SOLA supports the most descriptive version that includes 'All Attributes' to describe the schema items.

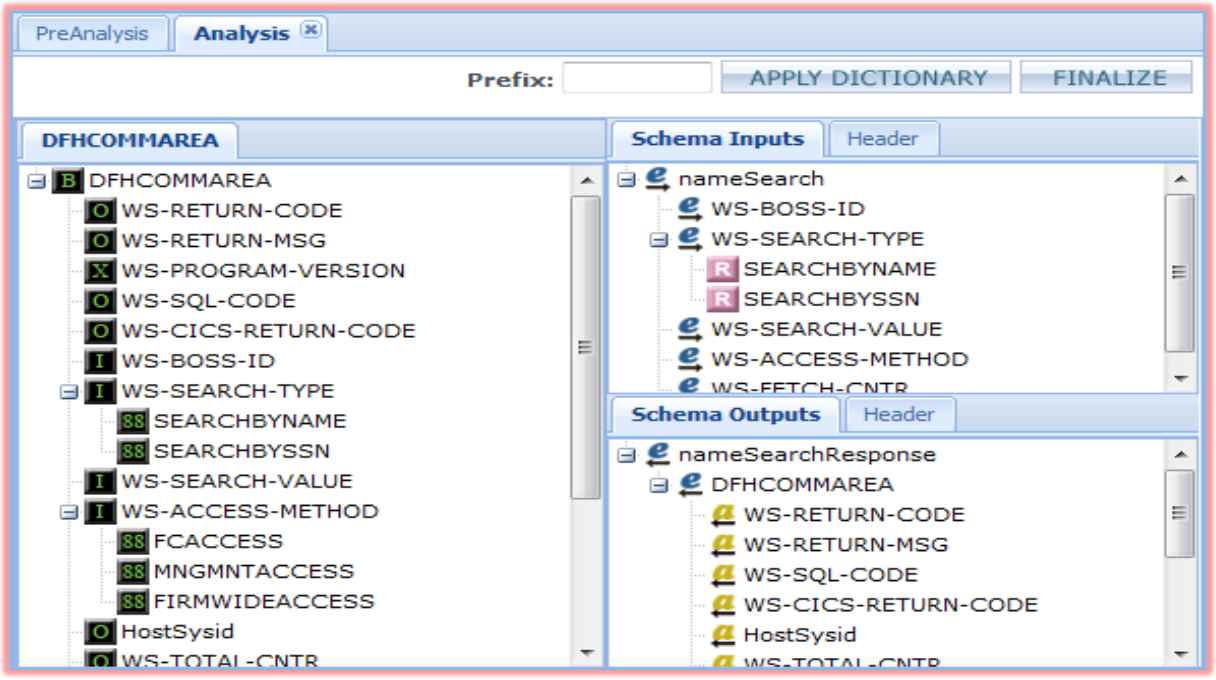
- **Target Namespace:** the URI of the defined operation (method). This can be adjusted to match your company's namespace standards.

- **Template Dataset & Load Dataset:** these fields are used to tell SOLA where you want the template source and the assembled/link-edited template to be stored. The source of the template will be stored as a member in the Partitioned Data Set (PDS) named in the **Template Dataset** field. The SOLA Analyzer will automatically assemble and link-edit the template into the Load Library specified in the **Load Dataset** field.

  Note: Duplicate program and template names will cause the following error to be displayed:



Fill in the required fields, then click ANALYZE the button. This will display the Commarea Analyzer.



Using the Commarea Analyzer

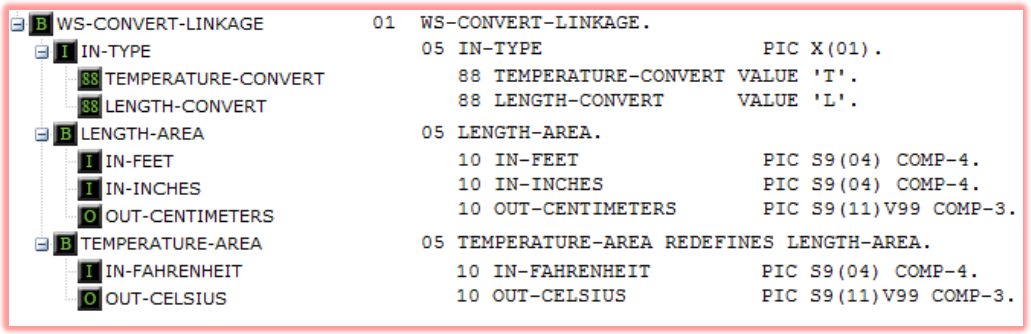
When you analyze a commarea method, what you are doing is creating a template (the runtime metadata), a test harness, directory entries and WSDL, which describes the interface to the program; the input and output fields. A web service is a way for a service consumer to call the legacy program; the consumer gives SOLA one or more inputs, which SOLA passes to the legacy program and receives a response, which it then passes to the consumer. What happens inside the legacy program is not relevant to SOLA or the consumer; either a correct response will be generated, or it will not. Therefore, even if a legacy program changes, as long as its interface remains the same, the web service does not need to change.

When analyzing a commarea program to create a method, you have to do the following:

- Isolate the functionality you want to use in your web service
- Determine what inputs the program needs to carry out that functionality
- Determine what outputs the program generates that are relevant to that functionality
- Add those inputs and outputs to the schema tree and configure them

The Legacy Tree

The legacy tree (located on the left in bottom-up analysis) represents the legacy data structure; the commarea. The schema tree (located on the right in bottom-up analysis) represents the WSDL that you are going to create. If you compare the Analyzer's legacy tree to the program's commarea you will see that they are a very close match.



Items displayed in the legacy tree are called Citems, and each Citem is represented by an icon. The following icons can appear in a legacy tree:

I	Input
O	Output
B	Input/Output (both)
X	Excluded
88	88 Level (enumeration)

In the legacy tree shown above, the fields are identified as being 88 levels, input, output or both. This is because this program was imported from a compile listing. When you import a compile listing SOLA evaluates the procedural code of the program (COBOL only) to determine which fields in the commarea are input, which are output, which are both input and output and which are excluded. This can save the user a lot of time, particularly when the commarea contains a lot of fields. If the program was imported from a copybook most of the items in the legacy tree would be represented with an X icon (excluded) and it would be up to you to determine which field is what.

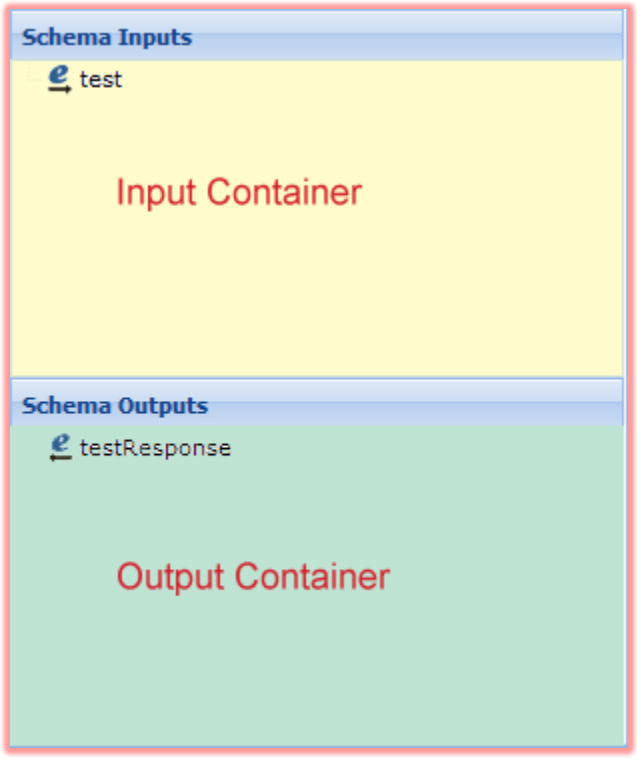
The icons represent the nature of the variable in the original commarea, not necessarily their value in the WSDL. Dragging an item into either the input or output portion of the schema tree will override the original variable type. Be careful when doing so, however, as you may render the web service inoperable. Regardless of how you arrange the WSDL, the legacy program still requires certain inputs and provides specific outputs.

When analyzing a bottom-up method (starting with copybook or compile listing), the legacy tree cannot be changed. If you were doing one of the other analysis types (top-down, meet-in-the-middle, etc.), then the opposite might be true.

Each item on both the legacy and schema trees is represented by a unique identifier. You can view the identifier by looking at an item's ID property in the property panel (properties are described later in this section).

The Schema Tree

The schema tree represents the structure of a WSDL, either one you are creating (in bottom-up analysis) or one you are working from (other analysis types). The tree is divided into two sections, Schema Inputs and Schema Outputs.



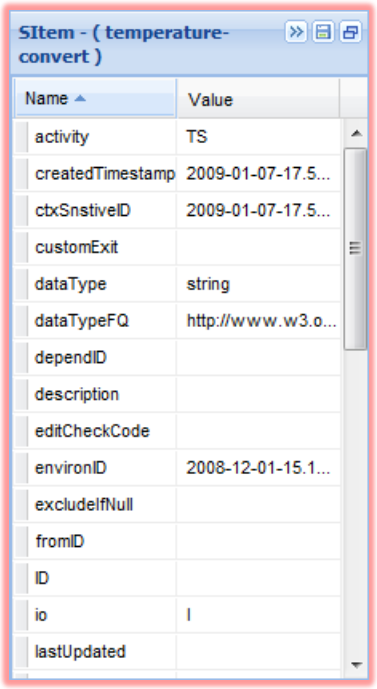
If you import from a compile listing, SOLA will attempt to populate these sections with the appropriate items from the legacy tree. However, if you import from a copybook, these sections will contain only upper level tree items (shown in the illustration on the left).

These items represent XML tags that will be present in the WSDL, and the inputs and outputs that the web service will use belong under (are child elements of) these items.

All items displayed in the Schema tree are called Sitems, and each Sitem is represented by an icon. The following icons appear in the schema tree:

	Element, input
	Element, output
	Attribute, input
	Attribute, output
	Restriction (enumeration)
	Default

Each item on both the legacy and schema trees is represented by a unique identifier. You can view the identifier by looking at an item's ID property in the **property panel.**



The Properties Panel

The properties panel is used extensively during analysis. It contains a host of properties for each item (variable) in the analyzer, both in the legacy tree and the schema tree. The panel serves two purposes, obtaining information and configuring tree items. You can use the panel to get such information as the value of an enumeration item or the minoccurs / maxoccurs value of an array item. Many of the properties can be changed, thereby allowing some fairly detailed configuration of tree items. The individual properties are detailed in the Commarea Analyzer Reference section on page 113.

Some of the property fields are text boxes, others are menus. The nature of any particular item will be apparent when you click on it. If the item is a text box, a blink cursor will appear and you will be able to make changes (unless that particular value cannot be changed). If the item is a menu, clicking on the item will reveal the menu with valid values for the item.

An enhancement to Analysis has been made in release 6.3.4; User can now change to any other compatible datatype during analysis and retain it, and if not compatible the datatype will be overridden with defaults. The default is for Numeric {short, int, long} based on precision, and the user can change to Decimal.

If you make changes to a property and want to save them, click the button. To reset all changes, click the button.

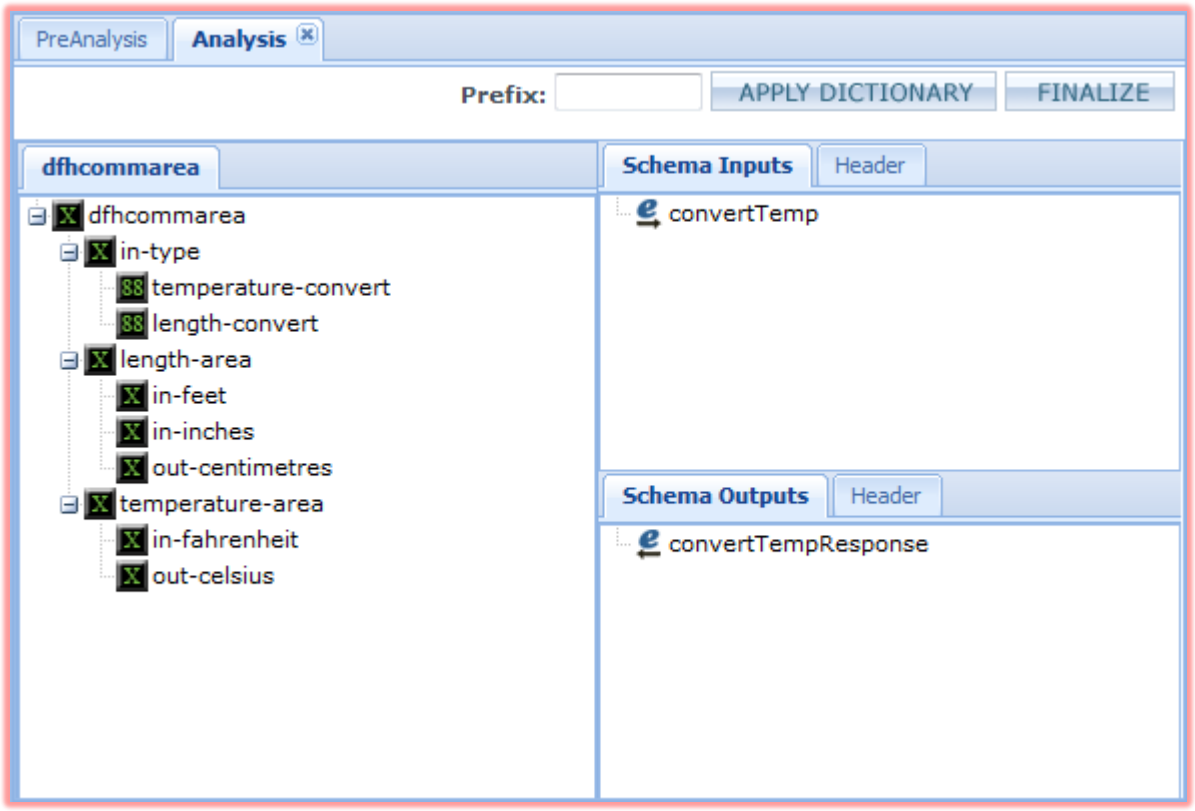
.

Analyzing the Method

To analyze a bottom-up inbound method, all you have to do is drag the input items you want from the legacy tree to the schema tree input container, then drag the output items you want from the legacy tree to the schema tree output container. Or, if you imported a compile listing, the input and output trees should already be populated. In that case, all you'll need to do is validate the schemas to be sure that they are correct and perhaps make some adjustments such as removing an item or adding a stop array (we'll review these later).

The more you know about COBOL or PL/I programs, the easier it is to perform an analysis. If you understand the program that was imported, and you know what you want the web service to do, analyzing a method is a very simple process, regardless of how complex the program may be.

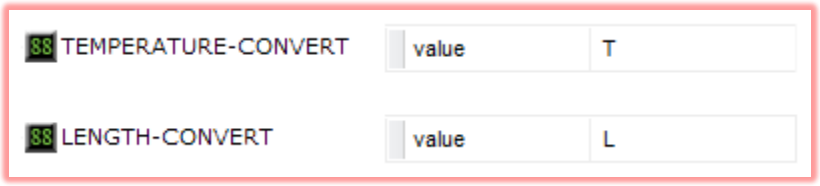
Let's take a look at a simple analysis.



This sample program (Convert) is part of the SOLA installation, and can be found in the SAMPLIB library that was shipped with SOLA. In this example, the program was imported from a copybook so SOLA couldn't identify the legacy tree items as input or output, so most of the items are represented by the X (excluded) icon. Since 88 levels are clearly identified in the code, these have been picked up by SOLA and are represented by the appropriate icon.

The program converts either length or temperature units from one system of measurement to another. It can convert feet or inches to centimeters or it can convert Fahrenheit to Celsius. The user specifies an input type (variable in-type), and the appropriate input, and the program returns the converted value.

Looking at the copybook (see page 59), you can see that the input variable in-type has two 88 levels, which are represented in the legacy tree. Clicking on each in turn, you can see that their values are shown in the Properties panel as being T and L.

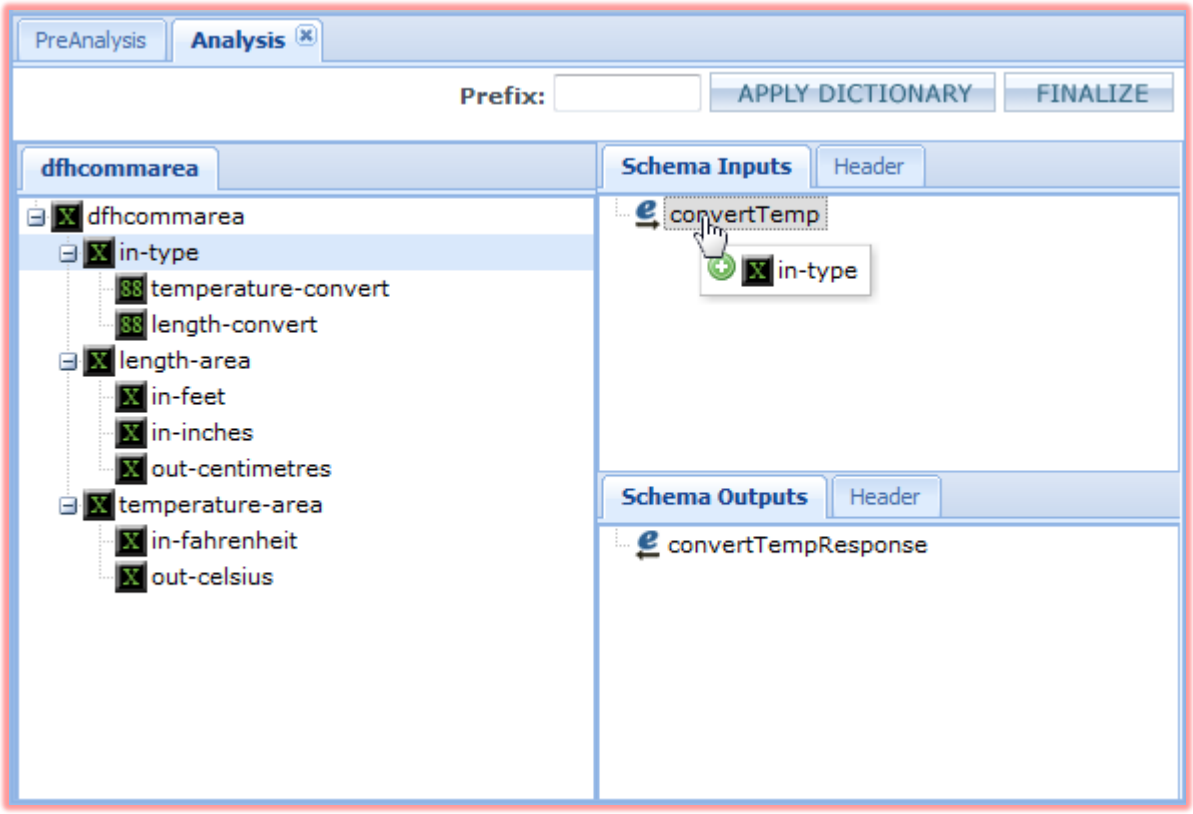


This indicates that in-type can have one of two values, T for temperature or L for length. The copybook also indicates that the temperature area redefines the length area, which means that you can only have one or the other, and the value of in-type determines which of those it is.

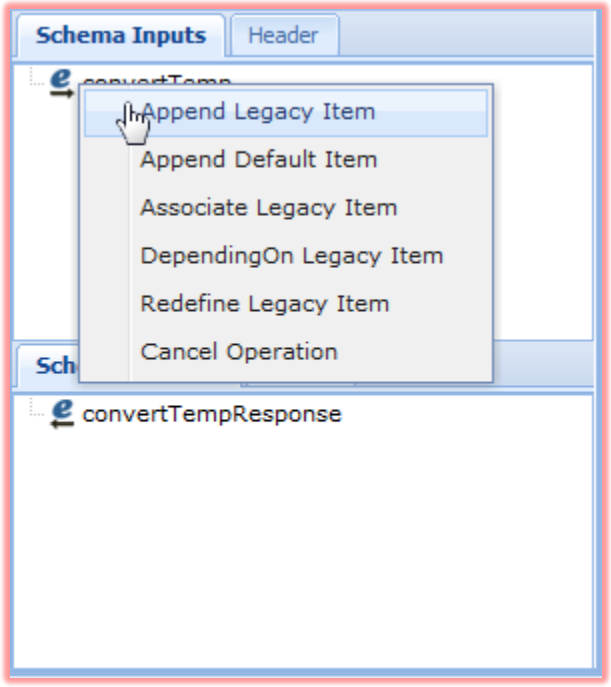
Redefines are common in legacy programs as they save memory, but they do limit us in terms of what kind of methods we can create. Had the temperature and length areas not been redefines, we could have created a single method that accepted in-type, IN-FEET, IN-INCHES and in-fahrenheit and performed the conversion based on in-type, returning OUT-CENTIMETERS and out-celsius. However, as those variable groups redefine each other, and only one can exist at any one time, this type of web service is impractical. In either case, it is more efficient to create two methods from this program, one to convert length, and one to convert temperature.

In this example, we are going to choose temperature. To convert temperature, the program will require two inputs; variable in-type with a fixed value of T, and in-fahrenheit. The program will then provide an output, out-celsius. We will need to configure our schema tree to reflect this.

First, drag in-type from the legacy tree and deposit it in your input area under the convertTemp node. This is like dragging a file from one folder into another. The destination folder, or in our case schema tree item, is "ConvertTemp" in the input container panel (Schema Inputs).



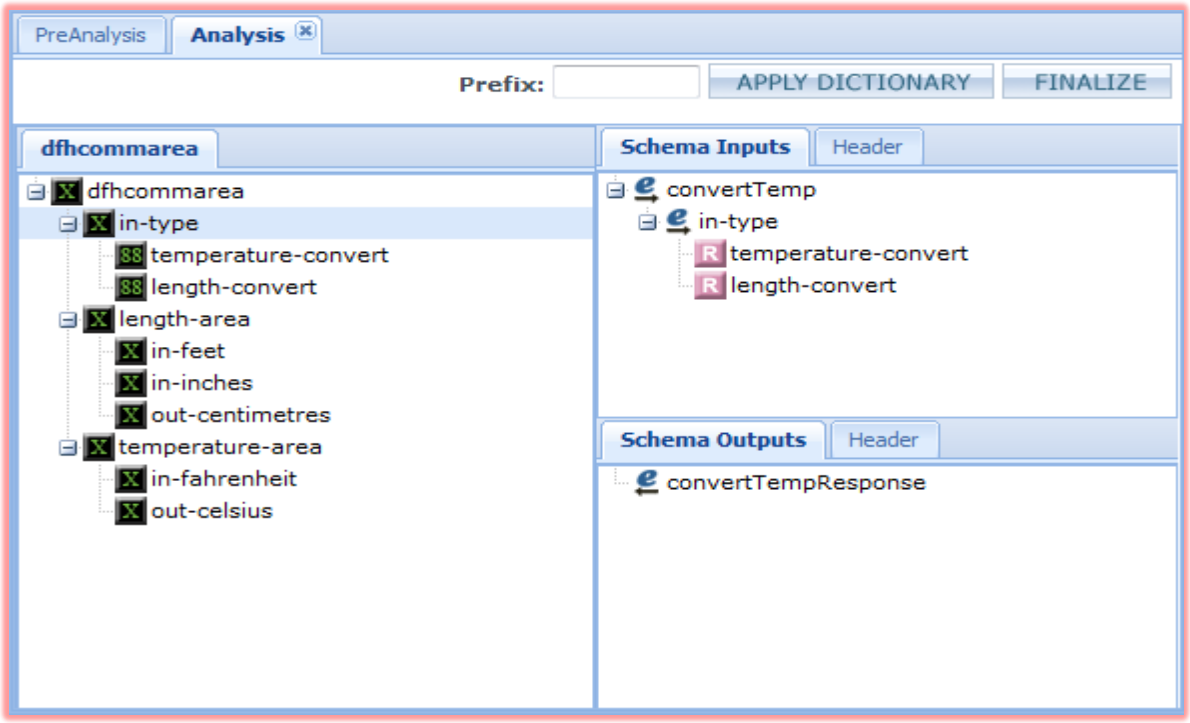
If you hold down the CTRL key when you click and drag, you will be presented with a menu of options.



The first choice, **Append Legacy Item**, is the default drag and drop operation (what happens when you drag and drop without using the CTRL key). The options not used in this example are described in the Commarea Analyzer Reference section on page 113.

To continue with the example, either do not use the CTRL key or select **Append Legacy Item** from the menu. The result will be the same.

Once you've moved in-type to the schema tree, the tree will look like this:

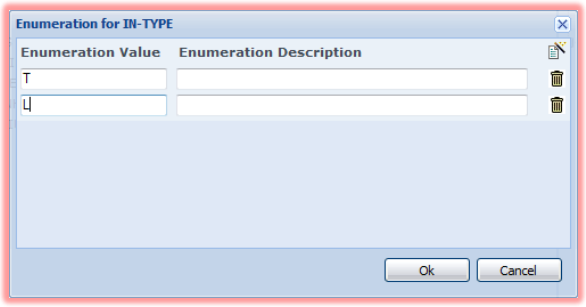


Note that if you click on in-type in the legacy tree and look in the properties panel for its IO type, it will be X (for excluded). However, once you drag in-type into the input section of the schema tree, it's IO type (in the schema tree only) will be set to I (for input).

in-type has two enumerations/restrictions R in the schema tree, which were derived from 88 levels in the COBOL data structure. This means it has two possible values, T or L. Since you have to pass a set value to the program, you should eliminate one of those values. To do so, use the Enumeration panel. You can use this panel to not only delete existing enumerations, but to add additional enumerations, change the values of existing enumerations and provide a description for each enumeration. This description will appear in the WSDL and may assist the distributed programmer in incorporating your WSDL into the front end user interface.

The following sample WSDL outlines the description as annotated documentation for each enumeration:

```
- <element name="WS-ACCESS-METHOD" minOccurs="0" maxOccurs="1">
  - <simpleType>
    - <restriction base="string">
      - <enumeration value="F">
        - <annotation>
            <documentation>FIRMWIDEACCESS</documentation>
          </annotation>
        </enumeration>
      - <enumeration value="N">
        - <annotation>
            <documentation>MNGMNTACCESS</documentation>
          </annotation>
        </enumeration>
      - <enumeration value="Y">
        - <annotation>
            <documentation>FCACCESS</documentation>
          </annotation>
        </enumeration>
      </restriction>
    </simpleType>
  </element>
```



To use the Enumeration panel, right-click in-type and select **Define Enumeration** from the pop-up menu. The enumeration panel contains all the existing enumerations of the item you clicked on. If there are no enumerations, the panel will be blank.

You can create new enumerations by clicking the icon and delete existing enumerations by clicking the icon next to the enumeration you wish to delete. To delete the L enumeration, click on its associated icon. This will remove the tree item LENGTH-CONVERT, leaving us with only one possible value for in-type, "T".

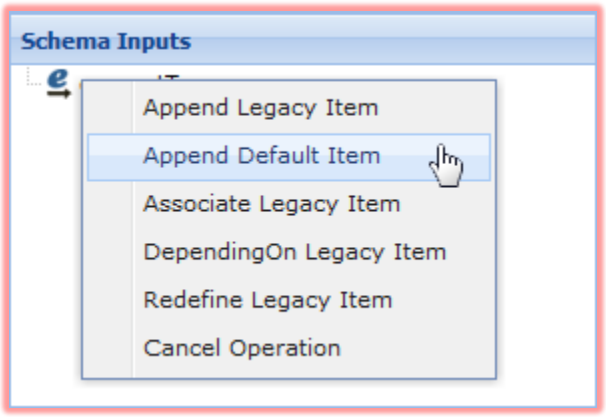
Deleting one of the enumerations will still require the web service consumer to send the value "T" for variable in-type, and while doing it this way is a good way to introduce you to how SOLA Developer handles enumerations, it is not a very efficient way to create this web service.

The general rule of default values vs. enumerations is as follows:

- **Enumerations:** restrict possible values but still require a value to be passed by the web service consumer. Most WSDL consumption tools will restrict that input so that the consumer can only enter a value that is present in the enumeration. There is no set limit for how many enumerations an item can have.

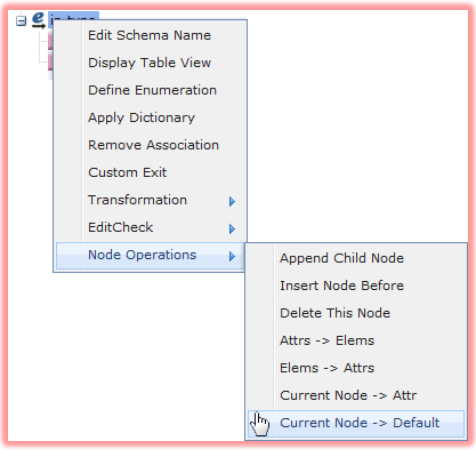
- **Default Values:** an item can have only one default value, and the web service will not require that the consumer pass this value, instead SOLA will pass this value to the legacy program.

The most efficient way to create this web service is to exclude in-type from the schema; there is no reason for the web service consumer, who will be accessing a program that converts only temperature to pass the conversion type. However, the legacy program, which converts both temperature and length, will still require a value to be passed for in-type. To accommodate both the consumer and the legacy program, you will need to assign a default value for in-type. Once a default value is assigned, the web service consumer will not see in-type in the schema, but SOLA will automatically pass the default value to the program.



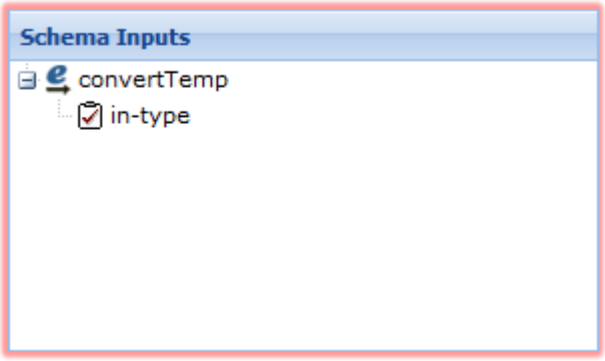
There are two ways to assign a default value. The first, used before the item is transferred to the schema tree, is to use the CTRL key when dragging and dropping. From the menu that pops up, chose **Append Default Item**.

If the item is already in the schema tree, right click on the item, and select Node Operations, followed by Current Node -> Default.

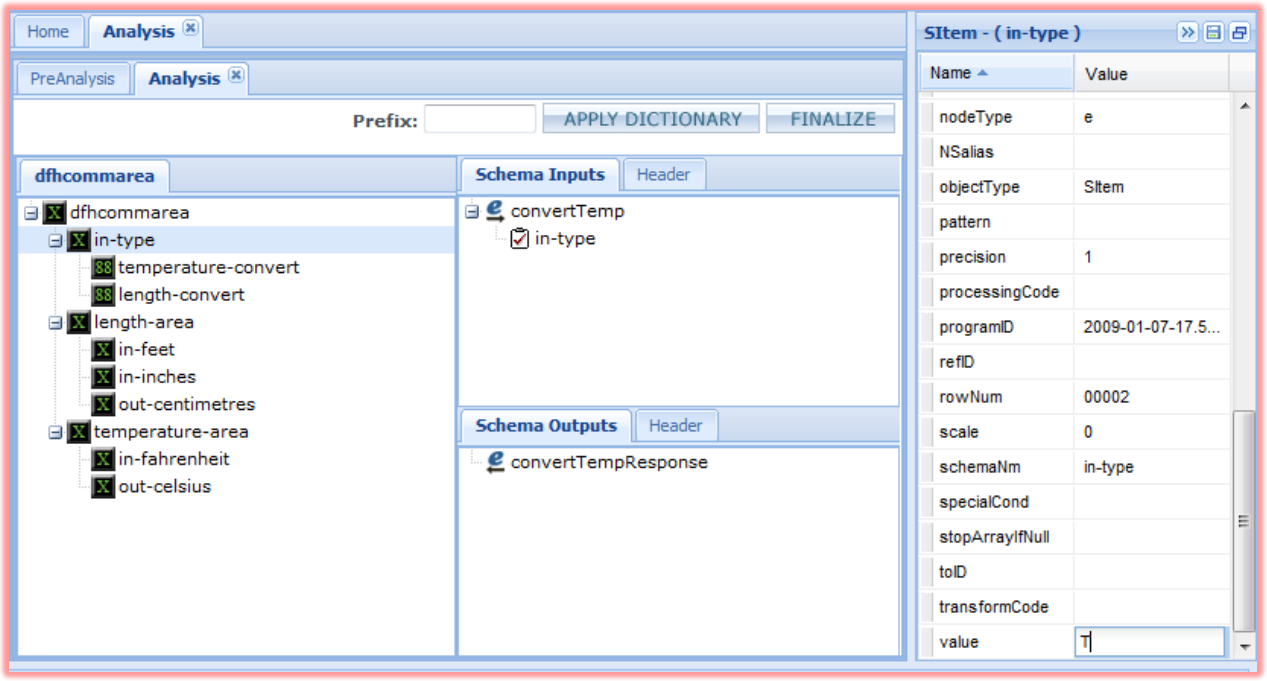


This will convert the selected node into a default node (a node with a set value that will not be present in the schema).

A default node cannot have enumerations, so if you placed it in the schema tree without using the CTRL key and then convert it to a default node, the enumerations will disappear.



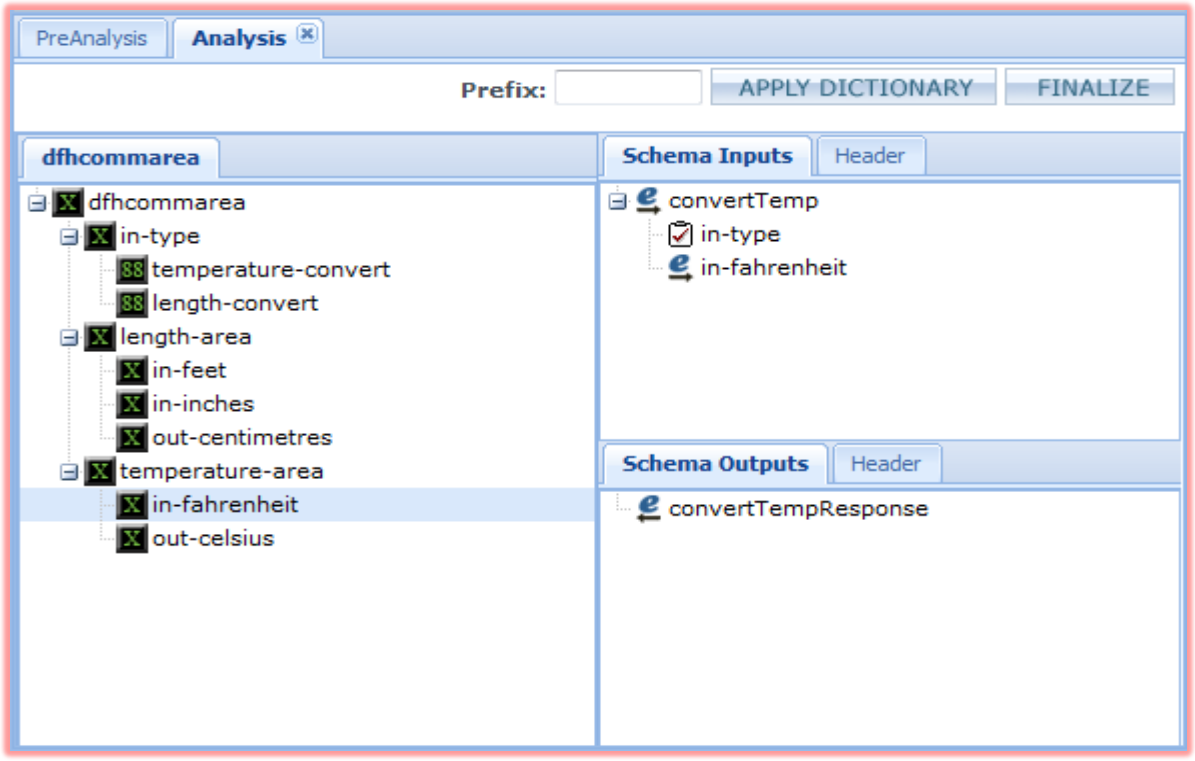
In either case, you will need to set the default node's default value in the properties panel. Select the default node, and then find the "value" field in the properties panel. Click anywhere in the empty value column for that field and enter a value of "T".



Click the button to save your changes. The default value for in-type has now been set and you are ready to proceed with the rest of the analysis.

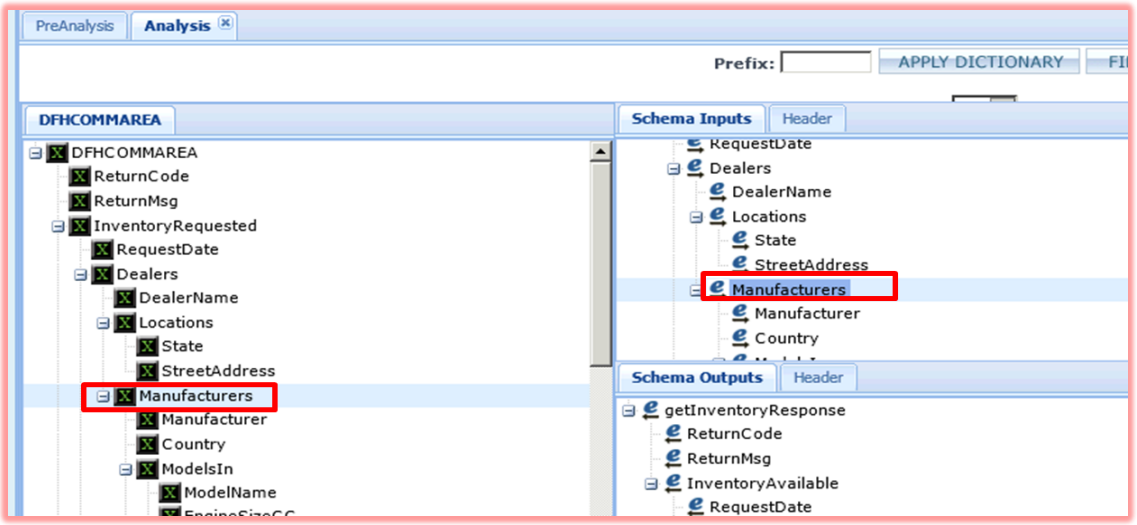
The next step is to drag the temperature conversion input, in-fahrenheit, to the input section. Drag it into the input container under the convertTemp node, just as you dragged in-type. All inputs go under ConvertTemp (or that same item with a different name in different programs), just as all output items go under convertTempResponse. Do not drag in-fahrenheit to in-type (so that it becomes a child of in-type), because that will indicate that in-type is a group and in-fahrenheit is a member of that group.

The schema tree will now look like this:



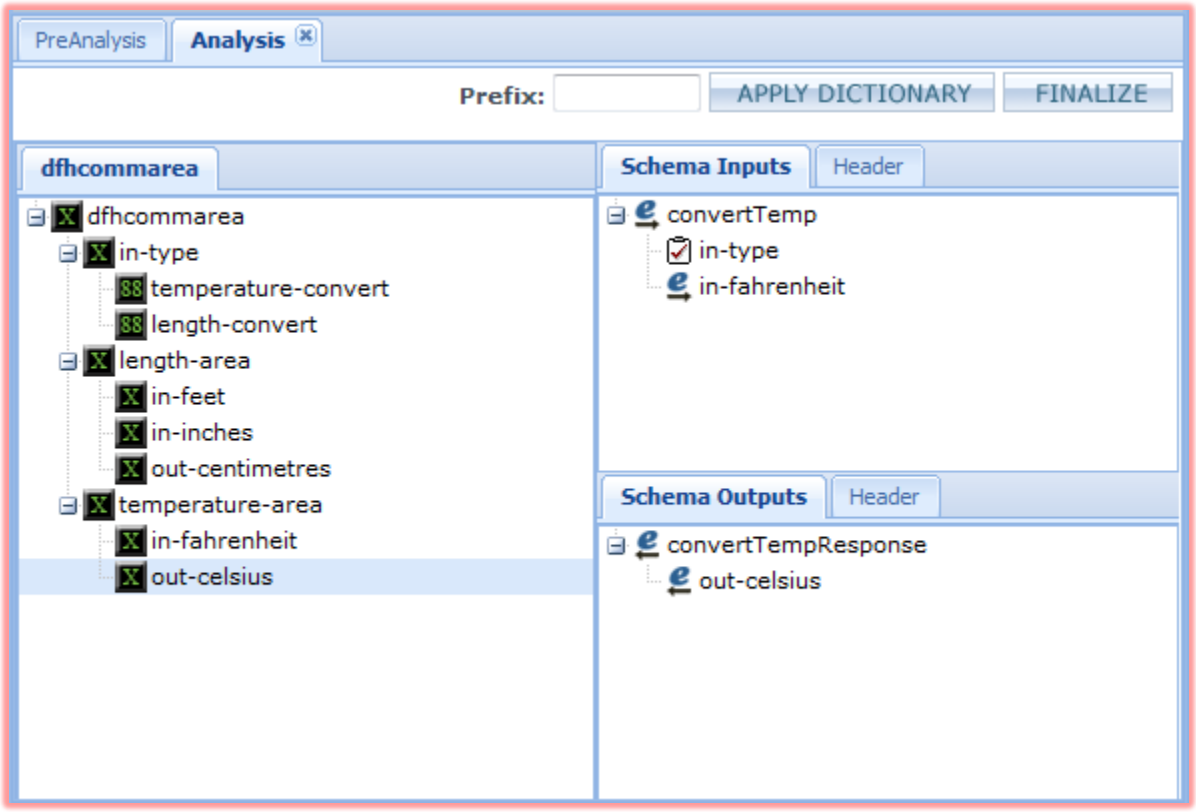
If you click on in-fahrenheit on the schema tree and look in the Properties panel, you will notice that it's IO type is also set to I (for input), just as in-type's was, as it was also dragged into the input section. The second thing to notice is that the value of in-fahrenheit is empty. This is because this input variable is not meant to have a fixed value like in-type is. By leaving the input value empty, the WSDL will indicate that the program expects this value to be provided by the web service consumer, and that the value can be anything (unrestricted).

To provide a visual 'schema-to-structure' mapping you can **click** on any schema field on the analysis screen and the corresponding legacy structure field will **flash** providing a visual cue to the developer about the field mapping.



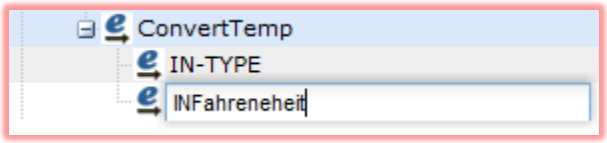
Now that we are finished working with the input section of our web service, it's time to configure the output section. In our simple web service, there is only one output variable, out-celsius. Drag out-celsius from the legacy tree to the output section of the schema tree, under ConvertTempResponse.

The schema tree should look like this:



Technically, our web service is finished. However, there is one more thing we can do to it in order to make our WSDL a bit more user friendly. SOLA Developer is equipped with a powerful global dictionary, and the principal function of the dictionary is to translate cryptic COBOL or PL/I names into human readable names. You can, of course, do this manually for each individual field.

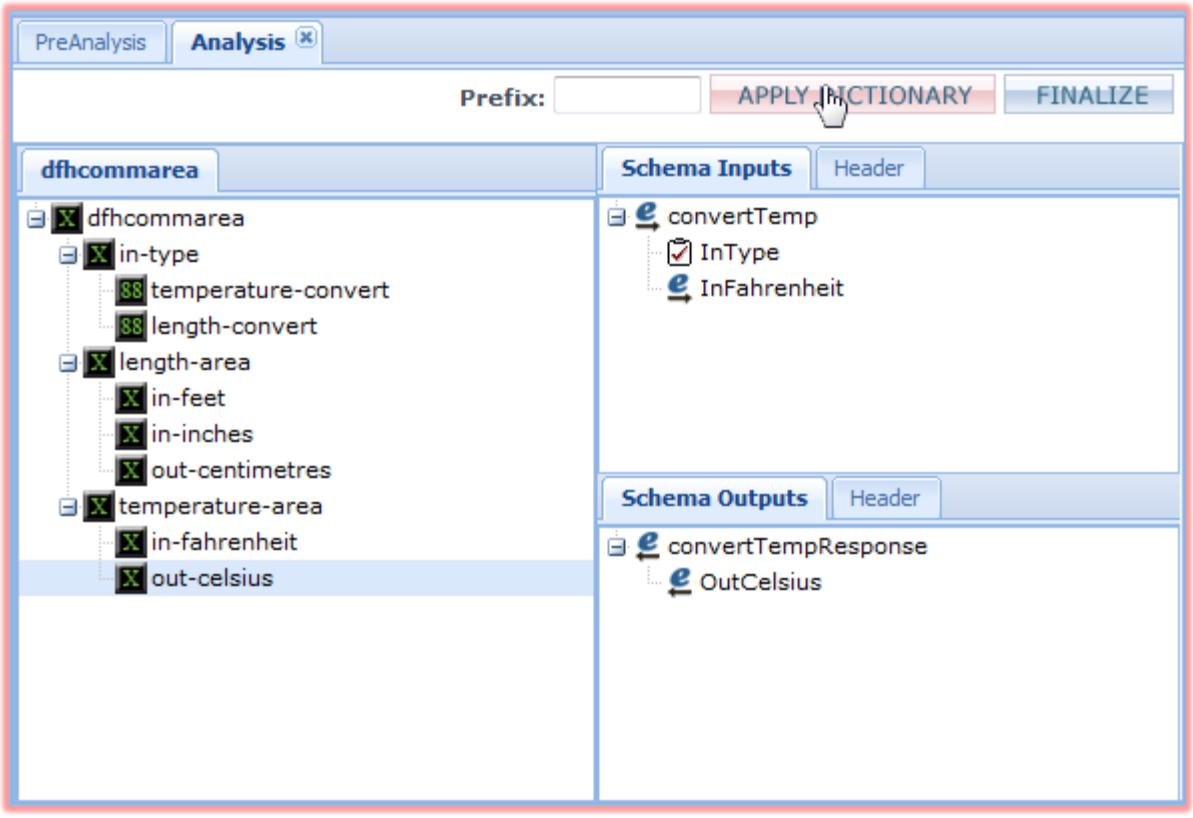
Double click on a field name to display a cursor (just like changing a file or folder name in Windows), then enter the item's new name.



It's much easier, however, using the SOLA dictionary, which allows you to click one button and change every name in the schema tree.

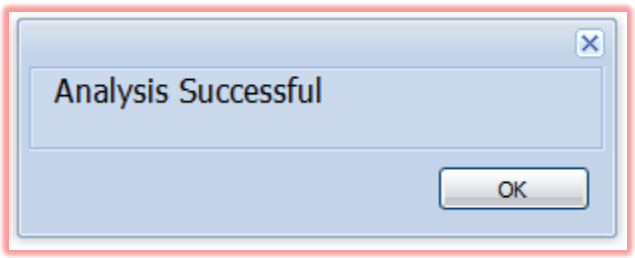
Click the 

button to translate all COBOL names in this web service to more user-friendly names.



Now our web service is fully configured, the field names have been translated and we're ready to conclude the analysis.

Click  to complete the analysis. You will be presented with a confirmation dialog.



The SOLA plugin has been enhanced to additionally generate SOLA Override fields in the WSDL as part of soap:Header. To generate the optional header fields and/or operation changes use these switches as follows:

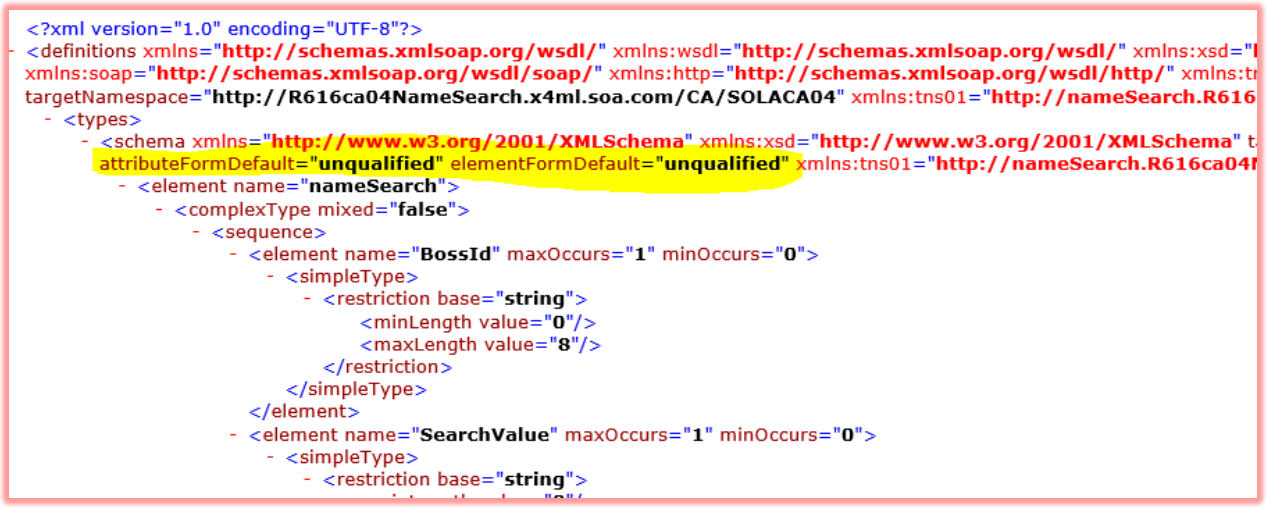
http://njstage-gx980:8618/sola/wsdl/TEST/Balaji_IE9_Testing/R618CA04/nameSearch?wsse=true

Using wsse=true will result in the Security elements being included in the soap header as in the following example:

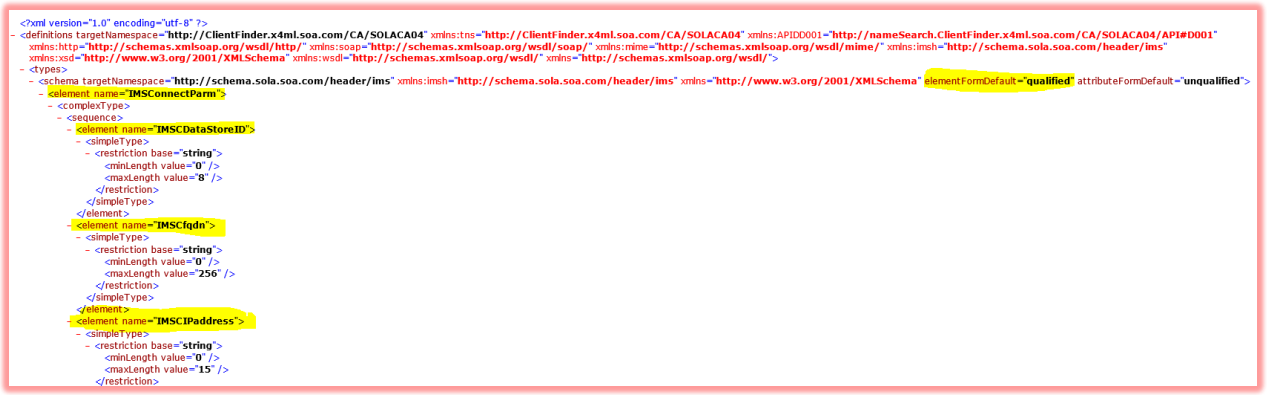
Note: excluding wsse=true is the equivalent of wsse=false and will not produce the security fields in the soap header.

Using unqualified=true will result in the elementFormDefault in the wsdl being unqualified for all operations within the schema as seen below:

http://njstage-gx980:1445/sola/wsdl/TEST/Balaji_619_Testing/R619CA04/nameSearch?unqualified=true



A sample WSDL using the imsheader=true in the URL will include an IMS Header in the schema (this is also discussed further in the IMS Plugin section):



Note: elementFormDefault is set to 'qualified' as the default when not coded on the URL.

A consumer of WSDL's generated by SOLA6 IDE (ProgramLevel or MethodLevel) can choose to use the following combination of parameters to adjust a wsdl:

http://{HOSTNAME}:{PORTNUMBER}/sola/wsdl/v1.1/byprogram/{Environ}/{Project}/{Program}?style=v5&terse=true&enums=documentation

The consumer can pick the style of WSDL(v5 or v6) as in the following example:

http://njstage-gx980:1445/sola/wsdl/v1.1/byprogram/TEST/.aaaaaaaDemo/SOLACA04/nameSearch?style=v5

In addition a consumer of a wsdl will be able to specify whether they want constraint meta-data for simple types within the WSDL as in the following example:

http://njstage-gx980:1445/sola/wsdl/v1.1/byprogram/TEST/.aaaaaaaDemo/SOLACA04/nameSearch?terse=true

And the consumer may also want to specify whether the enumeration definitions are represented as comments or not by specifying the parameter in the example below:

http://local.com:1445/sola/wsdl/v1.1/byprogram/TEST/QATEST-V6.3/QACA632P?enums=documentation

With enums=documentation - all enumerations will be displayed in the wsdl with their enumeration values as comments.

```
- <element name="WS-SEARCH-TYPE" maxOccurs="1" minOccurs="0">
   - <simpleType>
      - <restriction base="string">
            <!-- <enumeration value='N'> <annotation> <documentation>SEARCHBYNAME </documentation> </annotation></enumeration>
            </annotation></enumeration> -->
        </restriction>
    </simpleType>
  </element>
```

Without enums=documentation – all enumerations will be displayed in the wsdl as documentation:

```
<element name="WS-SEARCH-TYPE" maxOccurs="1" minOccurs="0">
  - <simpleType>
      - <restriction base="string">
          - <enumeration value="N">
              - <annotation>
                  <documentation>SEARCHBYNAME </documentation>
                </annotation>
            </enumeration>
          - <enumeration value="S">
              - <annotation>
                  <documentation>SEARCHBYSSN </documentation>
                </annotation>
            </enumeration>
        </restriction>
    </simpleType>
</element>
```

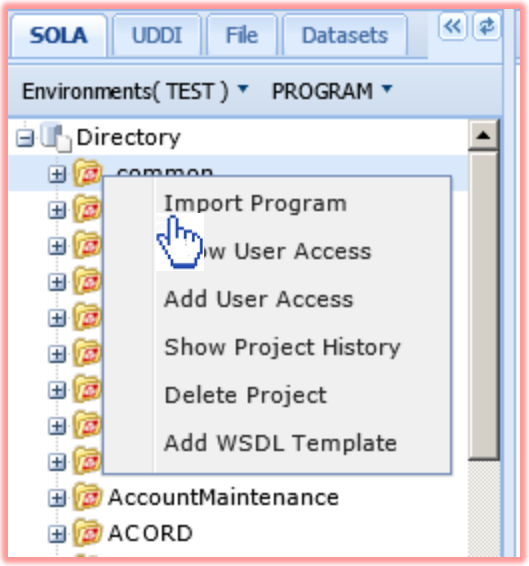
These instructions have only given you a bare overview of the functions and capabilities of the commarea analyzer. At the end of this chapter there's a reference guide that will help you to understand some of the more advanced features. Classroom training is also available from Akana Professional Services – SOLA Support Group.

Creating an Inbound Commarea Web Service from a WSDL – Top-Down

It is often the case that your corporate architects want you to conform to a standard service architecture, and they'll expect you to do so with a WSDL that they provide. Typically you won't have a mainframe program that exactly matches their requirements, and so you'll have to create one. SOLA supports this development paradigm by allowing you to generate a mainframe copybook (either COBOL or PL/I) from a WSDL file. You will write a program that uses that copybook as its interface, and your corporate architect will be able to call that program through SOLA.

Step 1 – Mainframe Preparation

The only mainframe preparation required with the top-down approach is for you to identify a PDS dataset for it to store the generated copybook (make sure that it is LRECL=80). You'll then write a program that incorporates that copybook, and you'll put it in a library that allows the SOLA runtime to call that program.



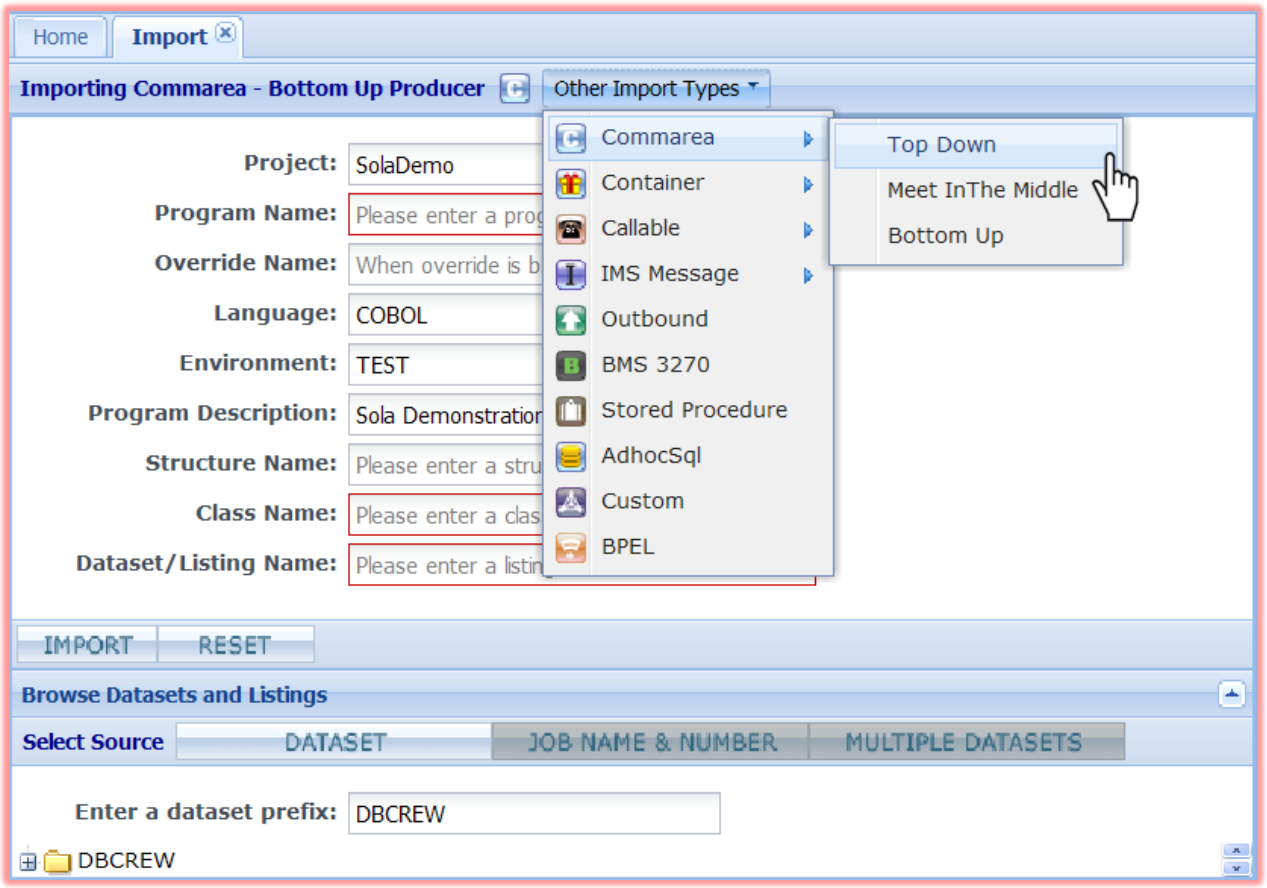
Step 2 – Importing the WSDL

Top down analysis is identical to that of inbound bottom-up Commarea programs (see page 45), but the import process is different, since you're importing a WSDL instead of a compile listing or copybook

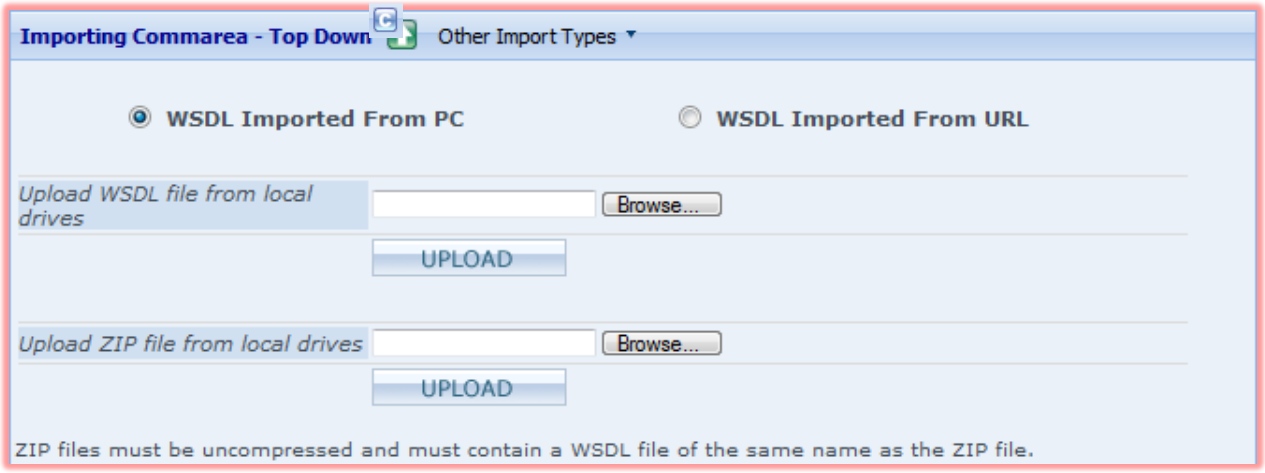
To get to the top down import panel, select the project you wish to import into and right-click it. From the pop-up menu, select **Import**. If you wish to import the program to a new project, first follow the steps for creating a new project on page 22.

After you select **Import**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

The default program type is commarea bottom-up, so use the **Other Import Types** menu to select commarea top down.



The Import panel will change to display the WSDL import panel.



This panel provides the means to import a WSDL file into SOLA from a location on the Internet/Intranet or on your local machine. To import a WSDL file from your local machine, either type the full path to the WSDL in the **Upload WSDL file from local drives** field or click the **Browse…** button to locate the WSDL using Windows Explorer.

To import a ZIP file, follow the same process described above, but use the **Upload ZIP file from local drives** field instead. SOLA doesn't support compressed ZIP files, so make sure the ZIP file you are uploading is uncompressed. This option is for the importing of WSDL files that utilize external references. The ZIP file must contain a WSDL file of the same name as the ZIP file and all files referenced by the WSDL.

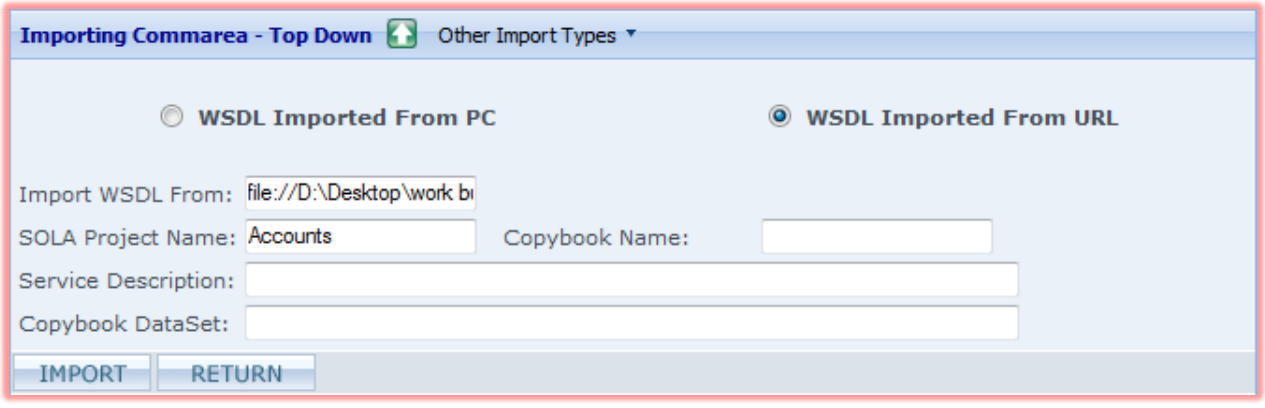
For example, if the ZIP file is called abc123.zip, then it must contain a WSDL file called abc123.wsdl and all files referenced by abc123.wsdl.

Once you have made your selection, click 

. Make sure you are clicking the correct Upload button (there are two).

If you want to upload the WSDL from a URL (internet/intranet) click 

. Either way, you will be taken to the following panel.



Fill in the fields as required:

- **Import WSDL From:** the address (URL) of the WSDL file being imported. If you chose to import from the local machine, the path of the local file will be displayed there. If you chose to upload from a URL, copy and paste (or manually enter) the URL into this field.
- **SOLA Project Name:** the name of the project under which the WSDL file will be imported. This is pre-populated and cannot be changed.
- **CopyBook Name:** the SOLA interface description for this call is placed in the specified Copybook Member.
- **Service Description:** a brief description of the service that will be created from the WSDL file.
- **Copybook DataSet:** the SOLA interface description for this call is placed in the Copybook Dataset you specify within the specified Copybook Member.

When you are ready to continue, click 

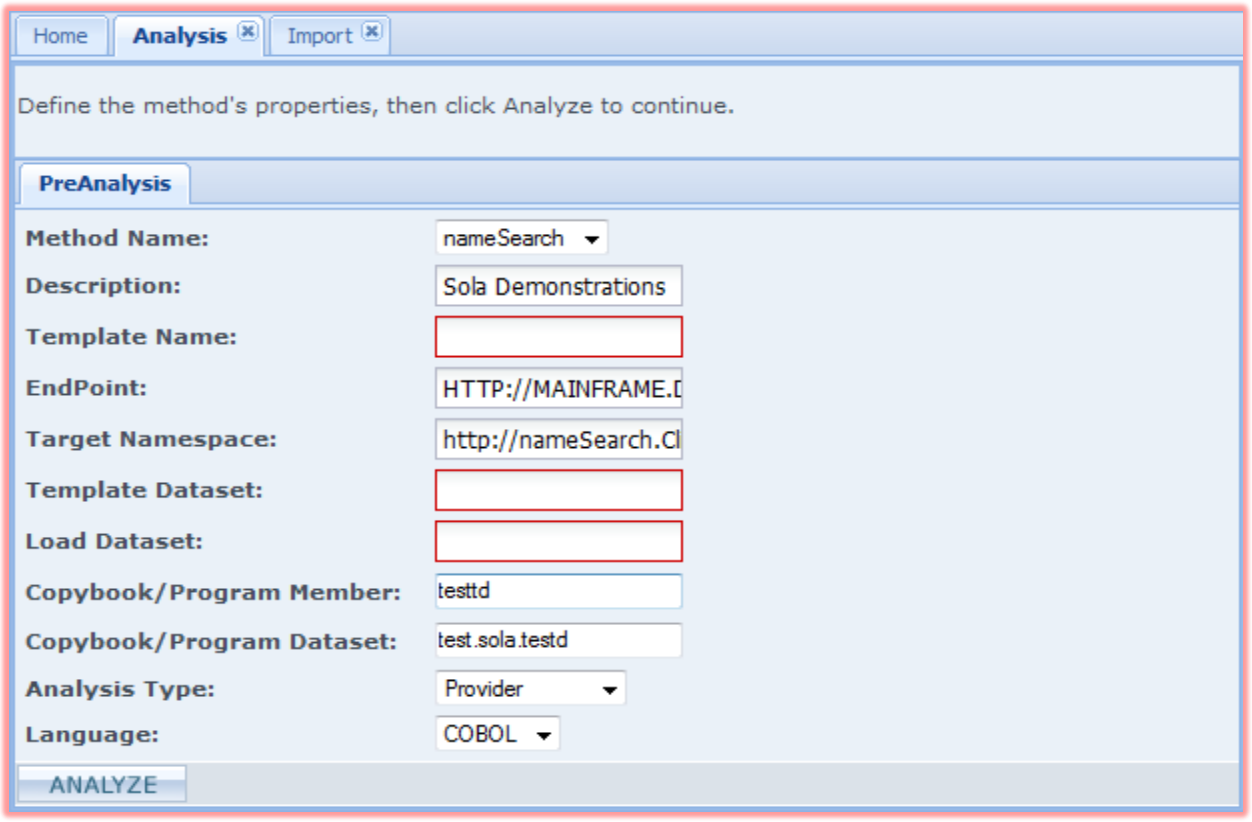
.

To return to the previous panel, click 

.

Step 3 – Analyzing the WSDL to Create the Copybook

Unlike bottom-up Commarea, clicking the Import button takes you right into analysis.



Fill in the required fields:

- **Method Name:** the name of the method being created.
- **Description:** a brief description of the method.
- **Template Name:** the name of the template (run-time metadata) that will be created for this method. The template name must be unique. The template tells SOLA how to facilitate communications between a legacy application and a distributed client or server. A template is an Assembler Data Only Load Module.
- **End Point:** the location of the SOLA SOAP server. Options will vary based on your installation.
- **Target Namespace:** the URI of the defined operation (Method).
- **Template Dataset & Load Dataset:** these fields are used to tell SOLA where you want the template source and the assembled and link-edited template to be stored. The source of the template will be stored as a member in the Partitioned Data Set (PDS)

named in the **Template Dataset** field. The SOLA Analyzer will automatically assemble and link-edit the template into the Load Library specified in the **Load Dataset** field.

- **Copybook/Program Member:** this is the name of the copybook that will be created as a result of the analysis.
- **Copybook/Program Dataset:** this is the location where the new copybook will be stored (PDS name).
- **Analysis Type:** options are "Consumer" and "Provider". Consumer means that the legacy program will act as a client to a distributed server (outbound) and Provider means that the legacy program will act as a server to a distributed client. For inbound commarea top-down analysis, the option must be set to Provider.
- **Language:** the language that will be used to create the copybook. Options are COBOL and PL/I.

When you have filled out the required fields, click the **ANALYZE** button.

The top down analyzer is identical to the bottom-up analyzer, except that the Schema tree is now on the left, and is the source of the analysis. The Legacy tree is on the right and is the target of the analysis.

The Legacy tree will represent SOLA's determination of how the copybook should be structured, based on the WSDL. You may, or may not, have to make adjustments.

To learn how to use the analyzer, read the commarea bottom-up example (page 56) and the commarea reference section (page 113).

Note: The provided WSDL must contain a SOLA compliant soap action as seen in the example below. Only include the codepage if other than the default codepage 37 is used.

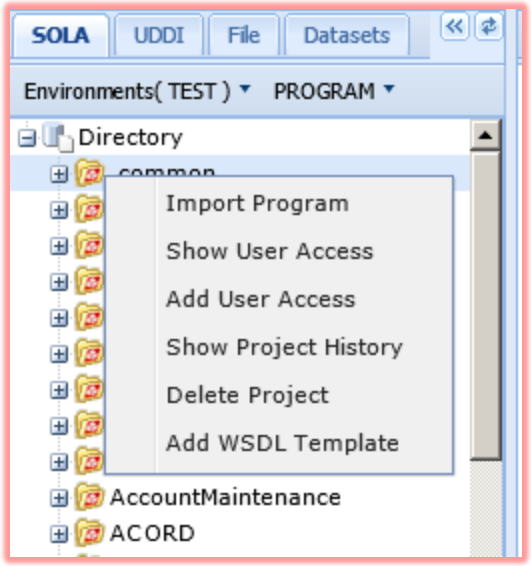
```
- <binding name="CLASS_QACA991PBindingName" type="tns:CLASS_QACA991PPortTypeName">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  - <operation name="QACA991M">
      <soap:operation style="document" soapAction="/CA/QACA99P/QACA991T/CCP:UTF-8/HCP:1140"/>
    - <input>
        <soap:body use="literal"/>
      </input>
```

Creating an Inbound Commarea Web Service from a WSDL and a Program– Meet-in-the-Middle

It is often the case that your corporate architects want you to conform to a standard service architecture, and they'll expect you to do so with a WSDL that they provide. When you have a program that fairly closely matches their requirements you can merge the WSDL and the program using SOLA's Meet-in-the-Middle approach, allowing your existing program to interact using the WSDL service definition.

Step 1 – Mainframe Preparation

The only mainframe preparation required with the meet-in-the-middle approach is for you to identify a copybook that describes the program's interface (typically DFHCOMMAREA). By using SOLA's heuristic dictionary, SOLA will map the WSDL fields to the copybook fields and will create a template containing those mapping rules.



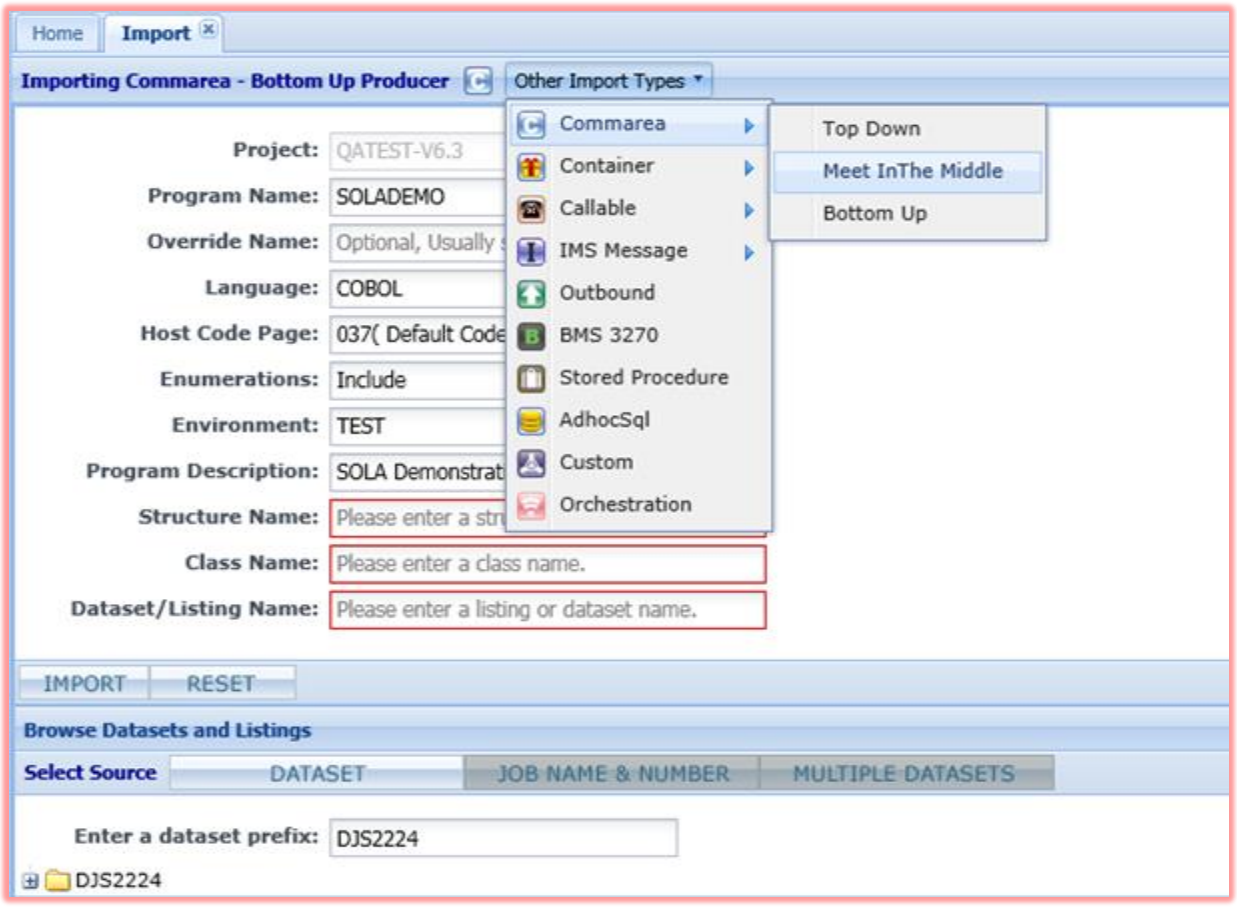
Step 2 – Importing the WSDL

Meet-in-the-middle analysis is similar to that of top-down analysis (see page76), but the import process is different, since you're importing a WSDL and a copybook

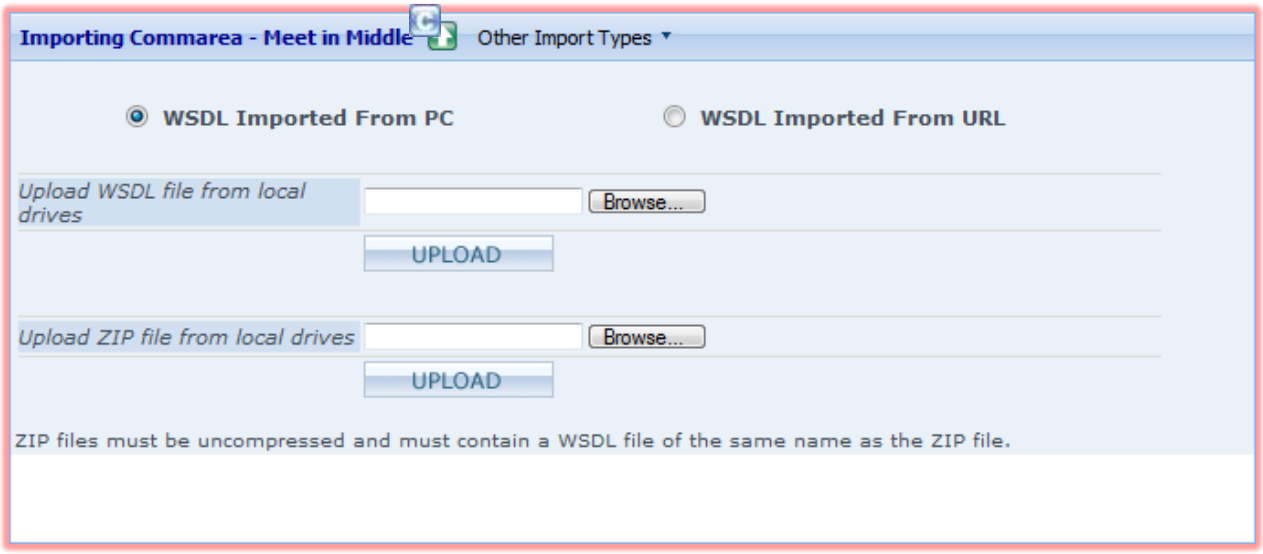
To get to the meet-in-the-middle import panel, select the project you wish to import to and right-click it. From the pop-up menu, select **Import**. If you wish to import the program to a new project, first follow the steps for creating a new project on page 22.

After you select **Import**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

The default program type is commarea bottom-up, so use the **Other Import Types** menu to select commarea "Meet-in-the-Middle".



The Import panel will change to display the WSDL import panel.



This panel provides the means to import a WSDL file into SOLA from a location on the Internet/Intranet or on your local machine. To import a WSDL file from your local machine, either type the full path to the WSDL in the **Upload WSDL file from local drives** field or click the **Browse…** button to locate the WSDL using Windows Explorer.

To import a ZIP file, follow the same process described above, but use the **Upload ZIP file from local drives** field instead. SOLA does not support compressed ZIP files, so make sure the ZIP file you are uploading is uncompressed. This option is for the importing of WSDL files that utilize external references. The ZIP file must contain a WSDL file of the same name as the ZIP file and all files referenced by the WSDL.

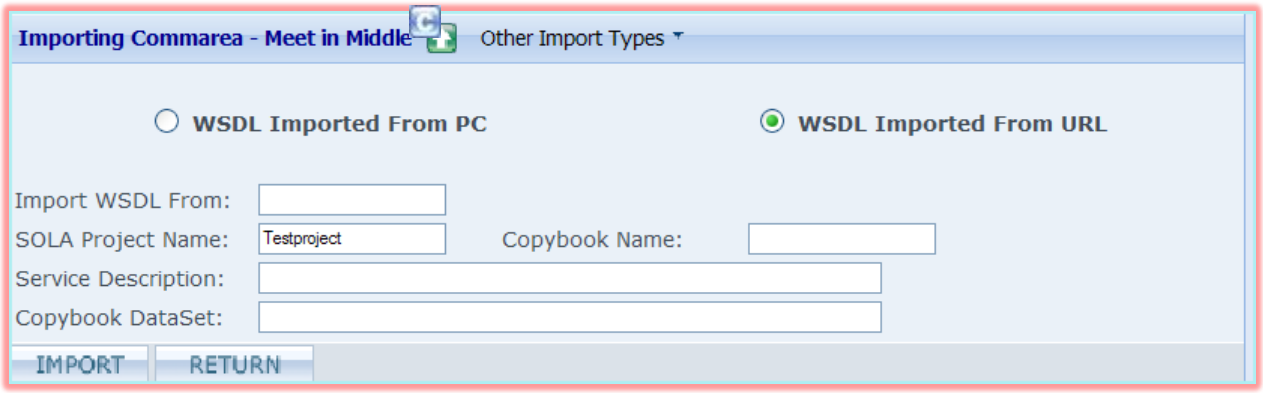
For example, if the ZIP file is called abc123.zip, then it must contain a WSDL file called abc123.wsdl and all files referenced by abc123.wsdl.

Once you have made your selection, click 

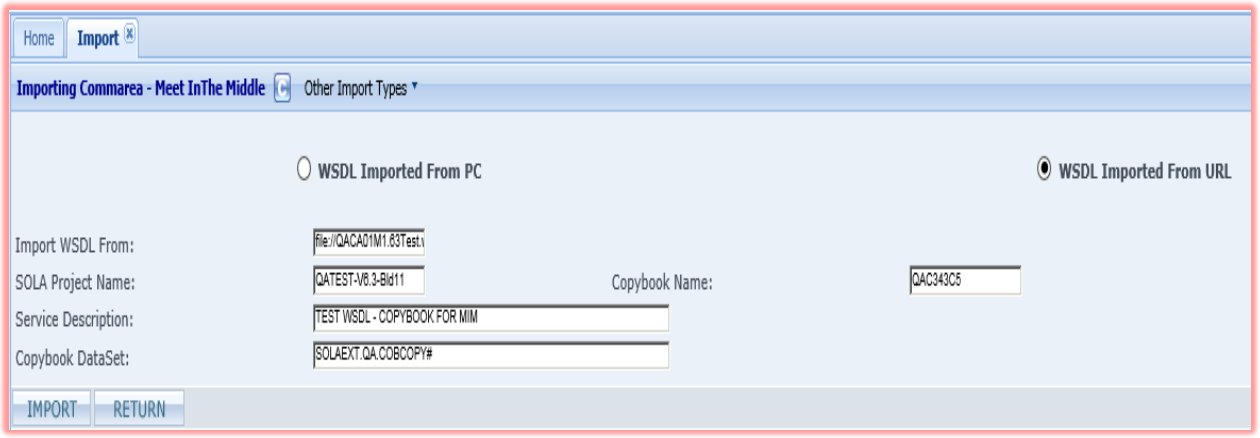
. Make sure you are clicking the correct Upload button (there are two).

If you want to upload the WSDL from a URL (internet/intranet) click 

. Either way, you will be taken to the following panel.



Fill in the fields as required:



- **Import WSDL From:** the address (URL) of the WSDL file being imported. If you chose to import from the local machine, the path of the local file will be displayed there. If you chose to upload from a URL, copy and paste (or manually enter) the URL into this field.

  Note: The provided WSDL must contain a SOLA compliant soap action as seen in the example below. Only include the codepage if other than the default codepage 37 is used.

```
- <binding name="CLASS_QACA991PBindingName" type="tns:CLASS_QACA991PPortTypeName">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  - <operation name="QACA991M">
      <soap:operation style="document" soapAction="/CA/QACA99P/QACA991T/CCP:UTF-8/HCP:1140"/>
    - <input>
        <soap:body use="literal"/>
      </input>
```

- **SOLA Project Name:** the name of the project under which the WSDL file will be imported. This is pre-populated and cannot be changed.

- **CopyBook Name:** specify the member name where SOLA is to find the interface definition (copybook) for your program.

- **Service Description:** a brief description of the service that will be created from the WSDL file.

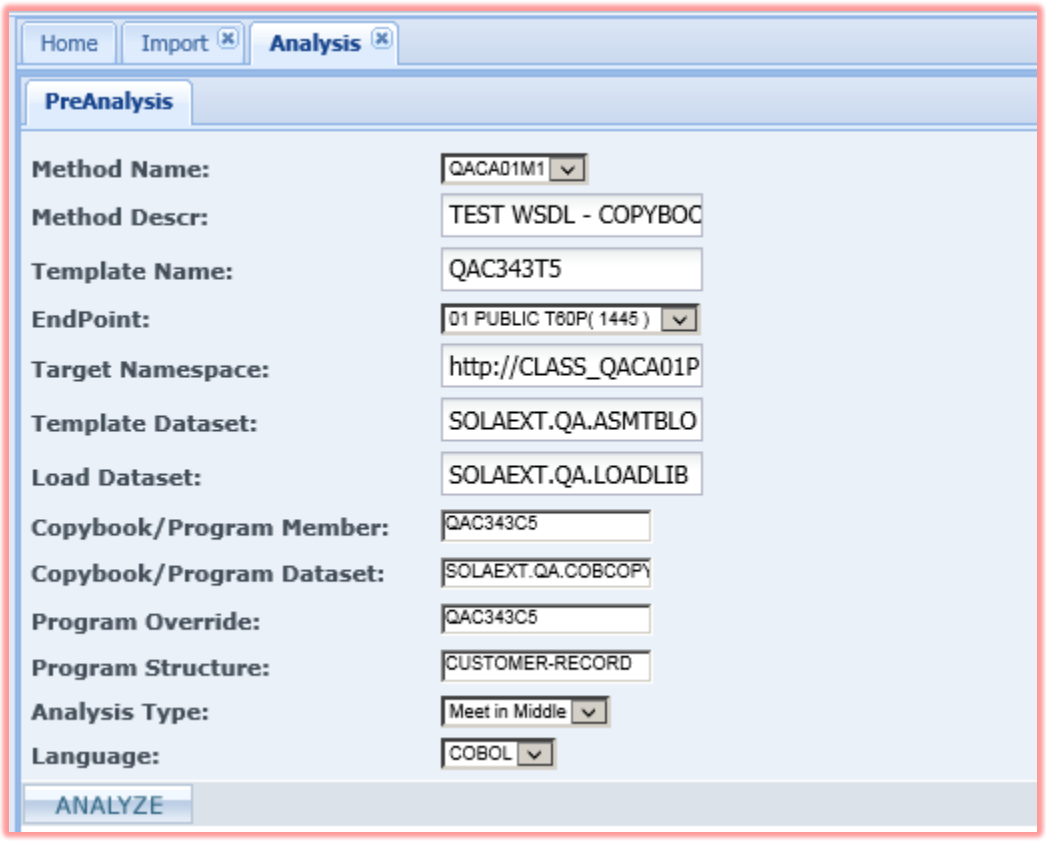
- **Copybook DataSet:** specify the Copybook DataSet (a PDS) where SOLA will find the interface definition (copybook) for your program.

When you are ready to continue, click IMPORT.

To return to the previous panel, click RETURN.

Step 3 – Matching the program to the WSDL

Unlike bottom-up commarea, clicking the Import button takes you into analysis.



Fill in the required fields:

- **Method Name:** choose the method name from the drop down list. The list is built from the operations in your WSDL.
- **Method Description:** a brief description of the method.
- **Template Name:** the name of the template (run-time metadata) that will be created for this method. The template name must be unique. The template tells SOLA how to facilitate the translation between SOAP and legacy data structures. A template is an Assembler Data Only Load Module.
- **End Point:** the location of the SOLA SOAP server. Options will vary based on your installation.
- **Target Namespace:** the URI of the defined operation (Method)
- **Template Dataset & Load Dataset:** these fields are used to tell SOLA where you want the template source and the assembled (load) and link-edited template to be stored. The source of the template will be stored as a member in the Partitioned Data Set (PDS)

named in the **Template Dataset** field. The SOLA Analyzer will automatically assemble and link-edit the template into the Load Library specified in the **Load Dataset** field.

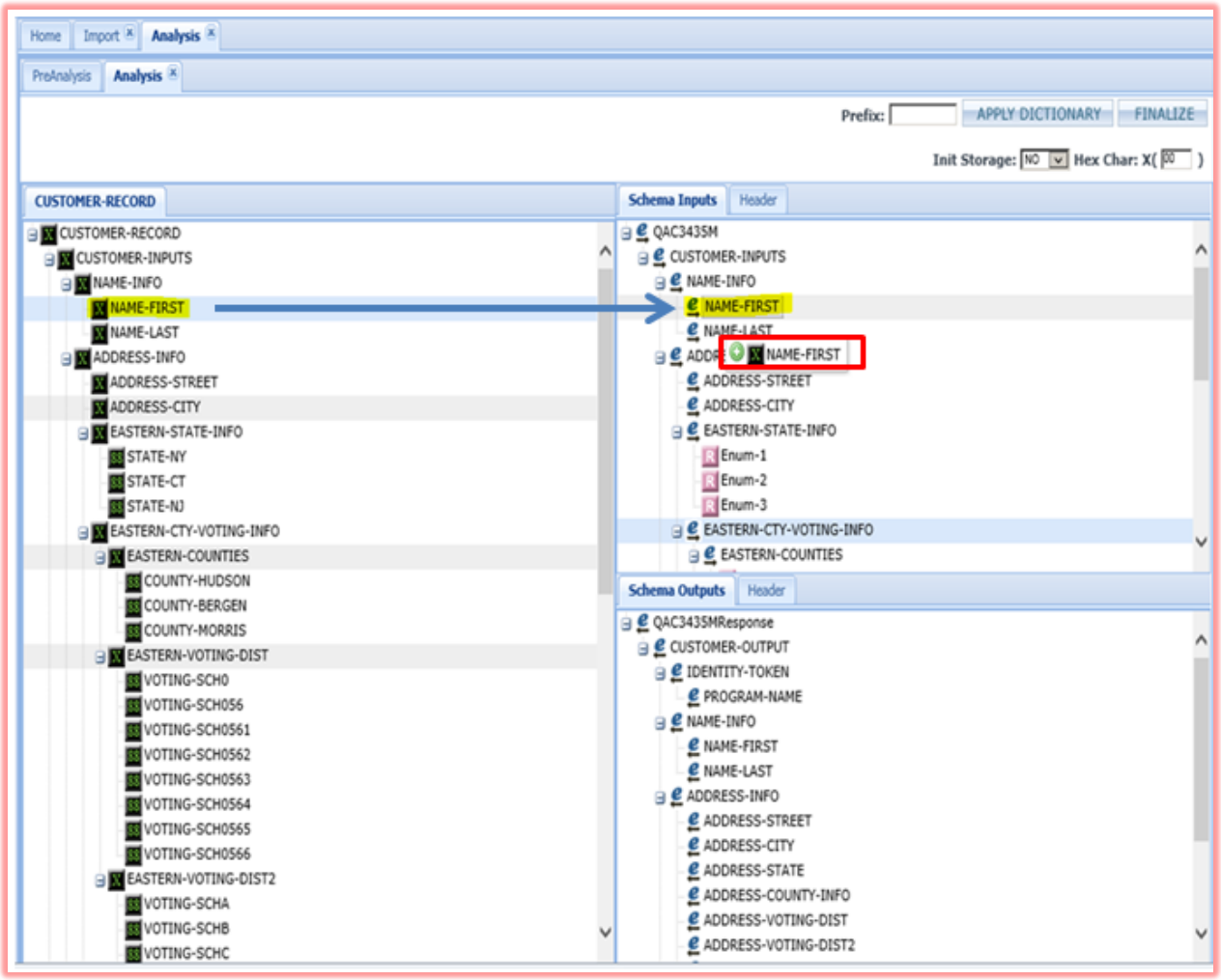
- **Copybook/Program Member:** this is the name of the copybook that describes your program interface.
- **Copybook/Program Dataset:** this is the location of your copybook.
- **Program Override:** The name of a target program to execute. Use this field when the target name differs from the program name (for example, when using the Program Name field for versioning).
- **Program Structure:** displays the COBOL or PL/I structure of the program (it's interface). For programs with multiple structures, this option shows all of the program's structures using tabs to navigate from one to the other.
- **Analysis Type:** options are "Consumer", "Provider" and "Meet in the Middle".
- **Language:** the language that will be used to create the copybook. Options are COBOL and PL/I.

When you have filled out the required fields, click the **ANALYZE** button.

The meet-in-the-middle analyzer is identical to the bottom-up analyzer, the Schema tree is on the right, and the Legacy tree is on the left.

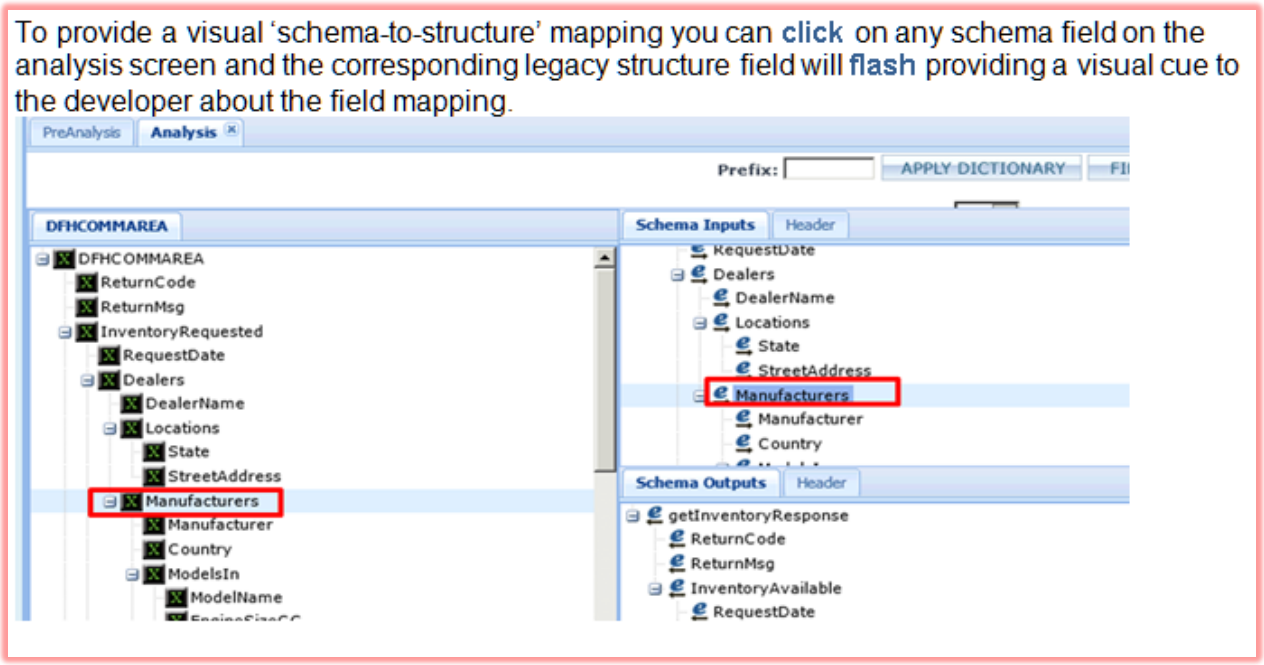
Your task is to match the elements in each tree, so that SOLA can perform the necessary transformations between the two structures. To do so, you must link every component in each tree to a matching item in the other tree by dragging and dropping each data element to the schema input and output.

In the following illustration we have begun by dragging each input or request element on the left legacy side of the workspace to its matching schema item on the right. You will continue with the output or response elements and linking them to the schema output also.



Note: The green circle with the element name will appear as you drag/drop the element to the correct schema element on the right side in the workspace.

Also, as you move along associating items to each other you will notice by clicking on a previously associated item in either side of the workspace, the item is automatically located and



Click FINALIZE to complete the analysis. You will be presented with a confirmation dialog indicating the analysis was successful.

To learn how to use the analyzer further, read the commarea bottom-up example (page 56) and the commarea reference section (page 113).

Creating an Outbound Web Service

SOLA is capable of bi-directional integration, meaning that the mainframe can be a server to a distributed client (as with all web services discussed previously), or the mainframe can be a client to a distributed server. This type of web service, where the legacy program is a client instead of a server, is called an "outbound" web service.

Like inbound top-down, you start with a WSDL and end up with a copybook. You will typically have to write a legacy program using that copybook, and invoke a SOLA program that will handle the outbound web service call and retrieve responses through the interface copybook.

Step 1 – Mainframe Preparation

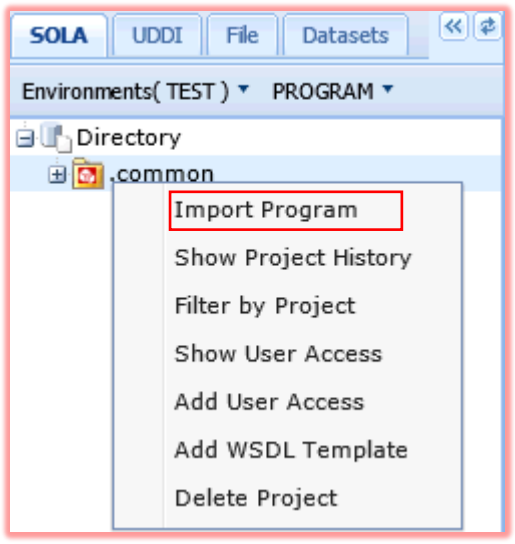
The only mainframe preparation required while creating an outbound web service is for you to provide SOLA with the names of three PDS files:

1. Copybook Dataset: A PDS to store the generated COPYBOOK.
2. Template Dataset: A PDS to store the generated metadata.
3. Load Dataset: A PDS to store the assembled and link-edited template (this dataset should be in your CICS region's DFHRPL concatenation).

After the analysis, you will need to write a program that incorporates that copybook and put it in a library that allows the SOLA runtime to call it.

Step 2 – Importing the WSDL

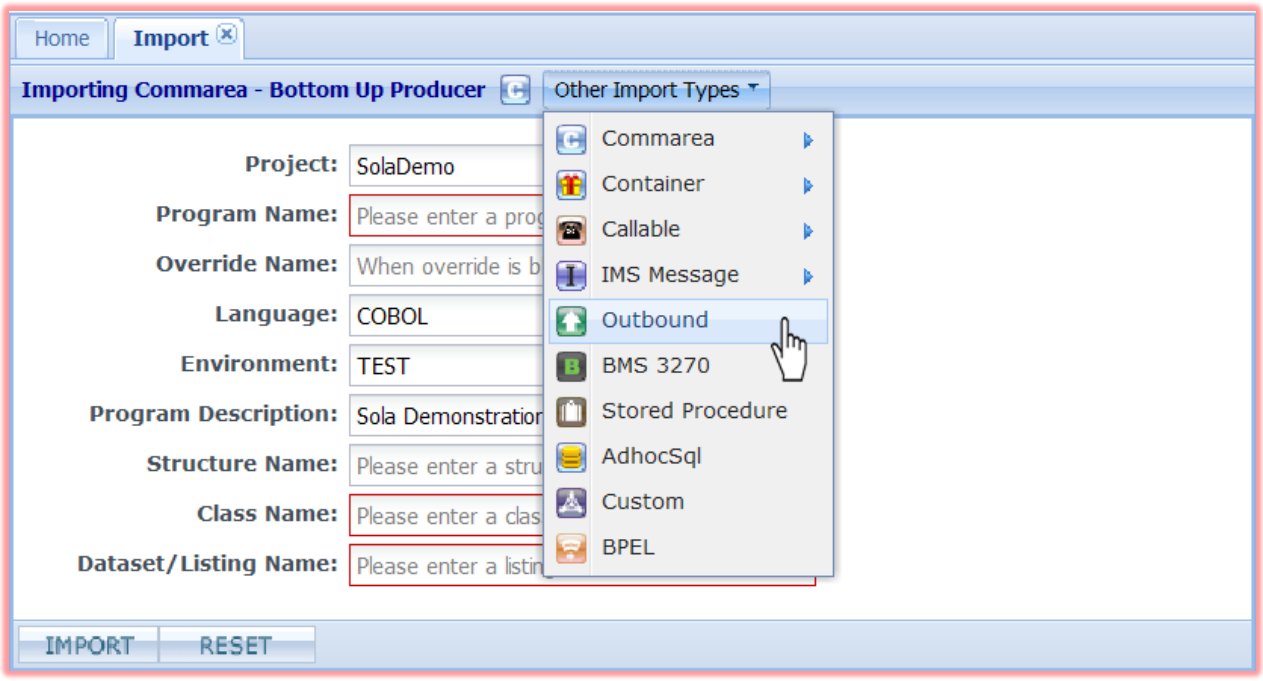
Outbound analysis is similar to that of inbound top-down program (see page 45).



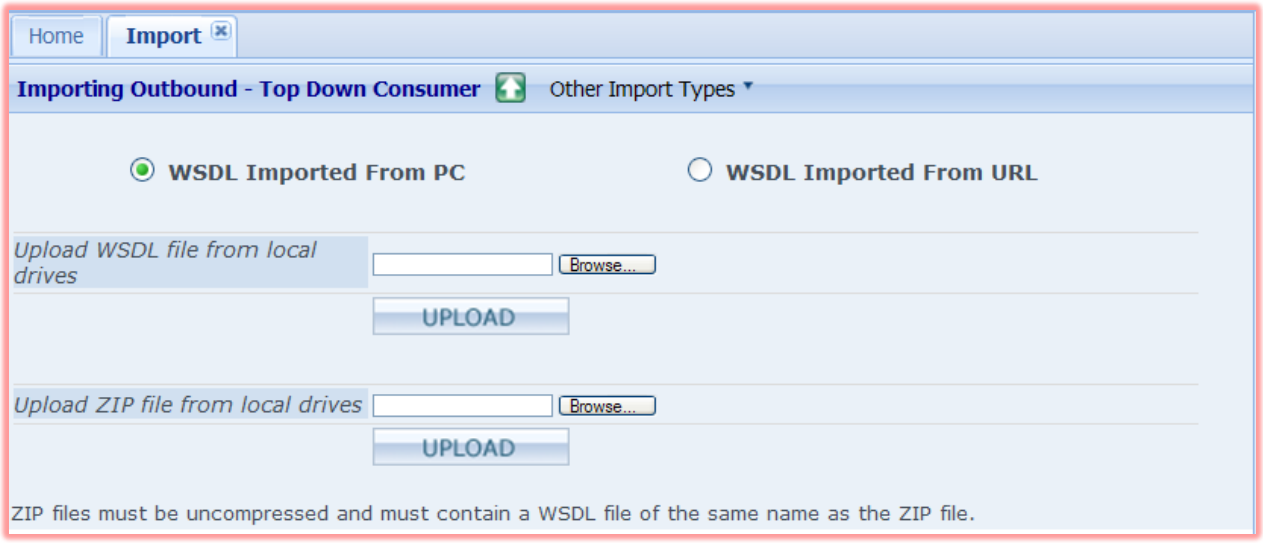
To get to the outbound import panel, select the project you wish to import to and right-click it. From the pop-up menu, select **Import Program**. If you wish to import the program to a new project, first follow the steps for creating a new project on page 22.

After you select **Import Program**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

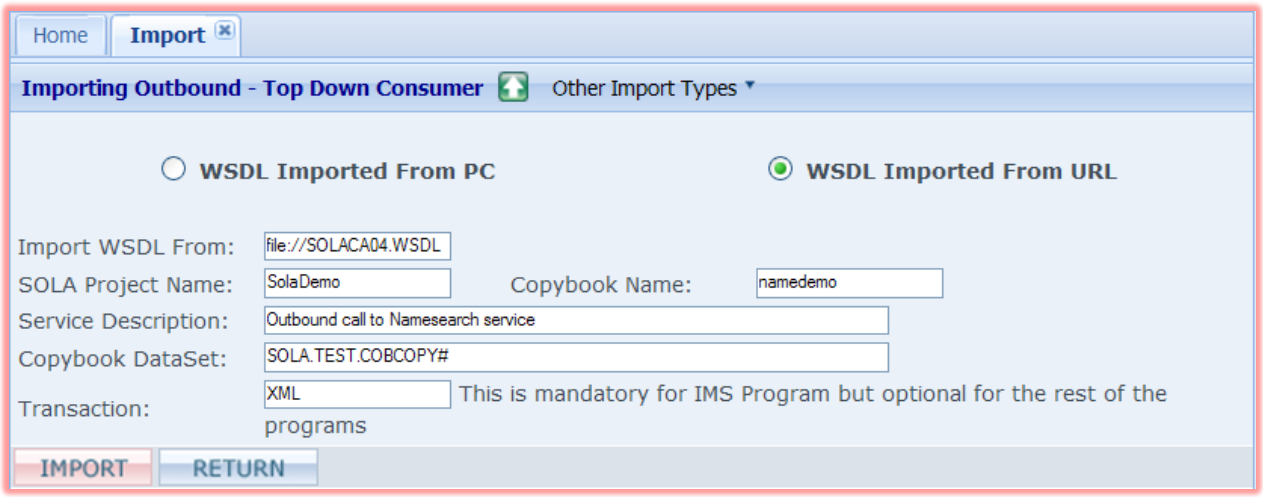
The default program type is "Commarea Bottom-Up", so use the **Other Import Types** menu to select Outbound.



The Import panel will change to display the WSDL import panel, as shown below. Enter the location of your WSDL file and click the Upload button. If the WSDL refers to included schemas you will need to zip them all together in an uncompressed zip file, and in that case you would enter the information in the "Upload Zip file from…" field.

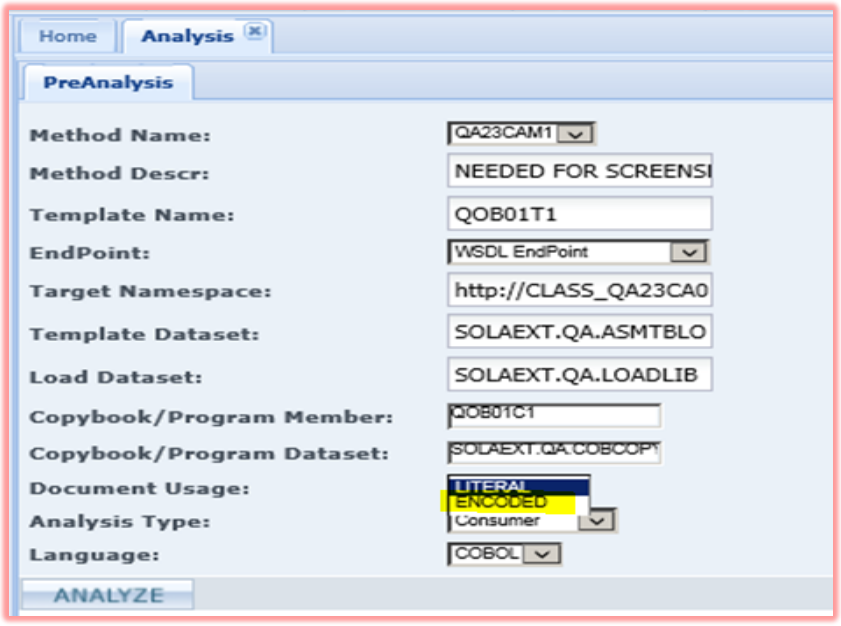


SOLA will read the WSDL file to determine its validity, and will then prompt you for the PDS and member name for the copybook that will be generated.



Enter the information requested and press the Import button to Import the WSDL.

At this point SOLA will determine the operations that the WSDL describes, and will present the PreAnalysis screen:



Choose the Operation (called **Method** by SOLA) and then enter the following non-case sensitive data:

Template Name: The PDS member where SOLA will store the metadata source.

Template Dataset: The PDS where SOLA will store the metadata source.

Load Dataset: The PDS where SOLA will store the runtime metadata.

Copybook/Program Member: The name of the mainframe copybook that will be created.

Copybook/Program Dataset: The name of the mainframe library/PDS where the copybook will be located.

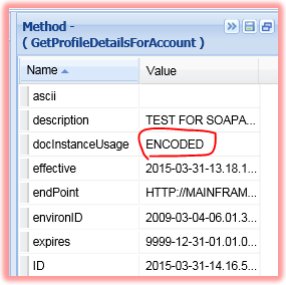
Document Usage:

Two choices defined as follows:

Selecting 'LITERAL' the request will only contain namespace referenced and no xsitype information.

Selecting 'ENCODED' the request will contain both namespace referenced and xsitype information.

The Document Usage choice is stored in the Property Panel on the right in property 'docInstanceUsage' :



Note: See example of WSDL containing Document Usage below:

```
- <wsdl:operation name="GetVisibilityInfo">
    <soap12:operation style="document" so
  - <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
  - <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  - <wsdl:fault name="MLIFServiceFaultFa
      <soap12:fault name="MLIFServiceF
    </wsdl:fault>
</wsdl:operation>
- <wsdl:operation name="GetProfileDetailsFo
    <soap12:operation style="document" so
  - <wsdl:input>
      <soap12:body use="encoded"/>
    </wsdl:input>
  - <wsdl:output>
      <soap12:body use="encoded"/>
    </wsdl:output>
```

In the following screen captures, Example 1: you will note a Request generated using Document Usage 'LITERAL' and in Example 2: a Request generated using Document Usage 'ENCODED':

Example 1:

```
<urn:GetProfileDetailsForAccountRequest Version="?" Resp
   <!--Optional:-->
   <urnl:BusinessPayload>
      <!--Optional:-->
      <urnl:AccountKeyList>
         <!--Zero or more repetitions:-->
         <urnl:AccountKeyData txType="?">
            <!--Optional:-->
            <urn2:AccountId>?</urn2:AccountId>
            <!--Optional:-->
            <urn2:AccountNumber>?</urn2:AccountNumber>
            <!--Optional:-->
```

Example 2:

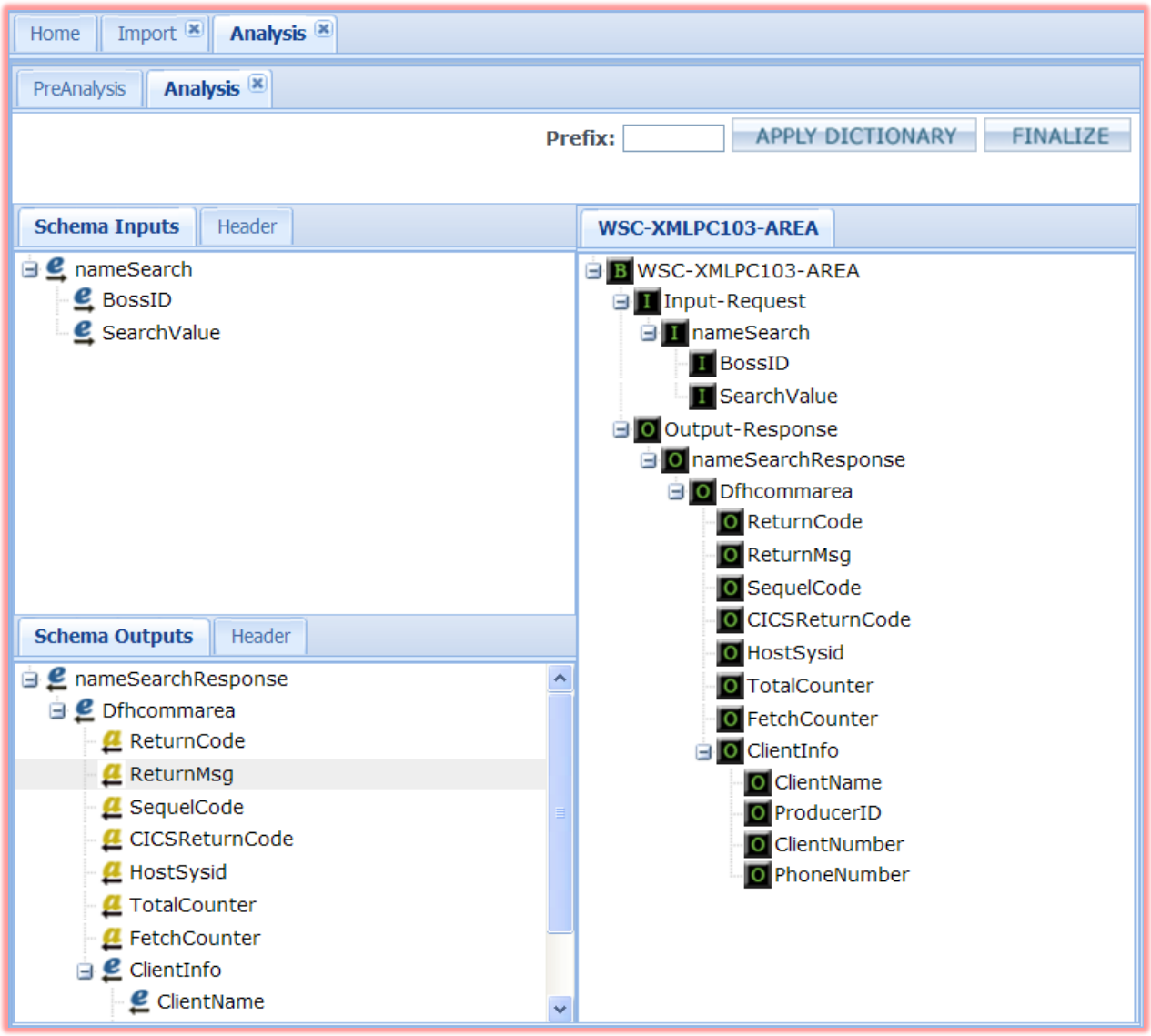
```
<BusinessPayload xsi:type="urn:GetProfileDetailsForAcctReqBusPayload
   <!--Optional:-->
   <AccountKeyList xsi:type="urn:AcctKeyListDataType">
      <!--Zero or more repetitions:-->
      <AccountKeyData xsi:type="urnl:AccountKeyDataType" txType="?"
         <!--Optional:-->
         <AccountId xsi:type="xsd:string">?</AccountId>
         <!--Optional:-->
         <AccountNumber xsi:type="xsd:string">?</AccountNumber>
         <!--Optional:-->
```

Analysis Type: When creating an outbound method Analysis Type must be set to "Consumer".

Language: Choose COBOL, PL/1 or Natural.

Once the information has been entered press the **Analyze** button to go to the Analysis screen.

In the example below we're analyzing the nameSearch operation that's described in the SOLACA04 WSDL file. This operation accepts two inputs and returns an array of data. The inputs are BossID and SearchValue and the output is return information and a bounded array of 300 Clients. The Analysis screen shows the input and output schemas on the left side of the screen and the equivalent mainframe structure on the right side.



Note: SOLA Outbound plugin can now consume WSDLs where element names are bigger than 50 characters.

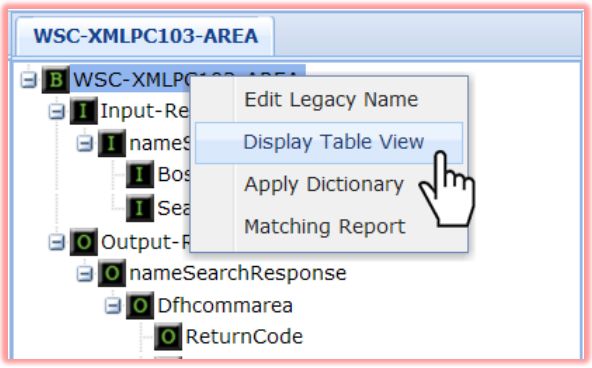
Clicking the Finalize button will create a mainframe copybook in the language you chose, a template (runtime metadata), directory (UDDI) entries and a test harness.

Sometimes the WSDL you're using isn't complete, and may only have the basic minimum information on the fields within it. For example, a field may be defined as a string datatype, but the WSDL doesn't say how long the field is. This is acceptable on distributed platforms, which use null terminated strings, but it isn't enough to build a mainframe copybook.

If no information is available in the WSDL, then by default SOLA will generate the legacy copybook as follows:

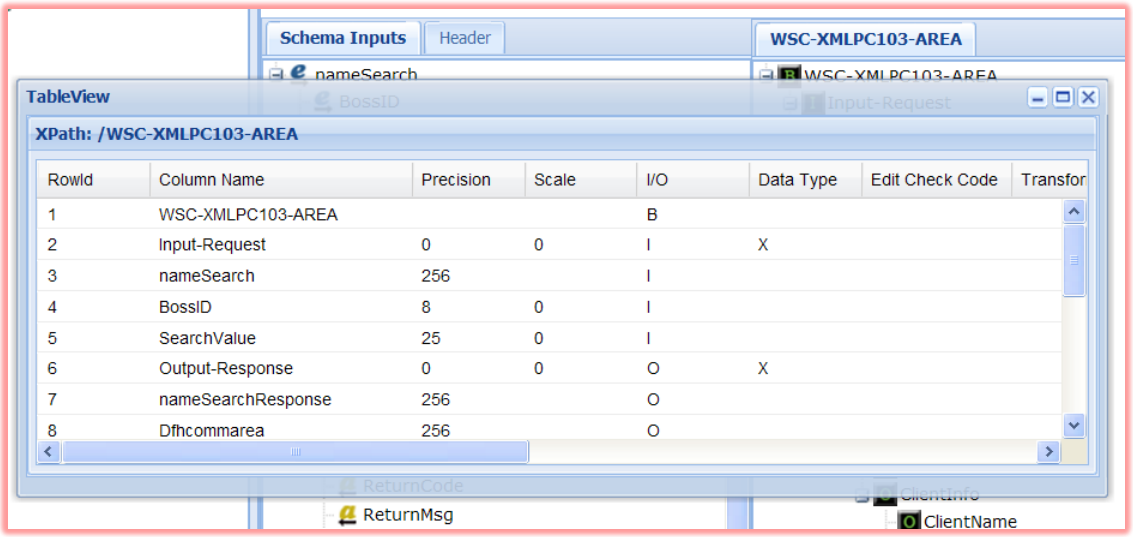
- String datatypes will be generated as PIC X(01).
- Double datatype will be generated as PIC S9(13)V9(4) COMP-3
- Unbounded arrays will be generated as OCCURS 9.
- All other datatypes will be generated according to their standard specifications, such as date, long, etc.

If the WSDL you're using lacks information then you'll have to tell SOLA about the fields, and you can do that by choosing "Display Table View" for the Legacy structure, as follows:



Right click on the Legacy Structure and choose "Display Table View" from the pop-up menu. This will display the fields in the Legacy structure in a columnar format.

SOLA will display a floating pop-up window where you'll be able to adjust the field metadata that will be generated into the legacy copybook.



RowId	Column Name	Precision	Scale	I/O	Data Type	Edit Check Code	Transfor
1	WSC-XMLPC103-AREA			B			
2	Input-Request	0	0	I	X		
3	nameSearch	256		I			
4	BossID	8	0	I			
5	SearchValue	25	0	I			
6	Output-Response	0	0	O	X		
7	nameSearchResponse	256		O			
8	Dfhcommarea	256		O			

Overtype the "Precision" field to modify the length of a character field. Modify the number of occurs by overtyping the Occurs field.

Note: With newer releases of SOLA the Commarea structure item (CItem) column named Data Type will no longer contain a value. This has been deprecated for commarea structures.

The Generated Interface Copybook

WARNING: Do not modify the generated copybook. Any modification can result in unpredictable behavior. If you need to make changes, do so by re-analyzing the method and make the necessary changes using the SOLA Analyzer.

WARNING: When including the copybook in your COBOL program DO NOT use the SYNC option as this will force full word alignment and possibly cause the address of copybook variables to no longer match the displacements recorded in the template metadata.

The copybook that SOLA generates has three sections, a protocol section, an input section and an output section. Because the copybook is generated you shouldn't modify it when overriding parameters, instead move the new values in your program prior to the call to SOLA. In the example above the protocol section of the COBOL copybook would look as follows:

```
* 01 WSC-XMLPC103-LINKAGE.
     02 WSC-INPUT-DATA.
        03 WSC-ENVIRONMENT                PIC X(01) VALUE 'O'.
           88 CICSINTERFACE               VALUE 'O' 'C'.
           88 CICSCONTAINER               VALUE 'C'.
           88 BATCHINTERFACE              VALUE 'B'.
        03 WSC-METHOD-NAME                PIC X(35)
     VALUE 'TestLargeComplexData'.
        03 WSC-TEMPLATE-NAME              PIC X(08)
     VALUE 'QWCT0601'.
        03 WSC-TRANSPORT-PROT             PIC X(01) VALUE 'H'.
           88 HTTPPROTOCOL                VALUE 'H'.
           88 MQPROTOCOL                  VALUE 'M'.
        03 WSC-ENDPOINT-DATA.
           04 WSC-SSL-IND                 PIC X(01) VALUE 'N'.
              88 USE-SSL                  VALUE 'Y'.
              88 AT-TLS                   VALUE 'A'.
              88 NO-SSL                   VALUE 'N'.
           04 WSC-NODE-1                  PIC 9(03) VALUE 000.
           04 WSC-NODE-2                  PIC 9(03) VALUE 000.
           04 WSC-NODE-3                  PIC 9(03) VALUE 000.
           04 WSC-NODE-4                  PIC 9(03) VALUE 000.
           04 WSC-PORT                    PIC 9(05) VALUE 80.
           04 WSC-FQDN                    PIC X(128)
     VALUE 'inside.test.principal.com'.
           04 WSC-FILE-PATH               PIC X(128)
     VALUE   '/SolaOutboundTesting/services/SolaOutboundTestingSer
-    'vice'.
           04 WSC-PROXY                   PIC X(122) VALUE SPACE.
           04 WSC-PROXY-PORT              PIC 9(05)  VALUE 0.
        03 WSC-MQ-MANAGER-DATA REDEFINES WSC-ENDPOINT-DATA.
           04 WSC-CONVERS-TYPE            PIC X(01).
              88 DATAGRAM                 VALUE 'D'.
              88 REQUESTREPLY             VALUE 'R'.
              88 REPLYTO                  VALUE 'T'.
           04 WSC-MANAGER-NAME            PIC X(48).
           04 WSC-QUEUE-NAME              PIC X(48).
           04 WSC-REPLY-TO-QUEUE          PIC X(48).
           04 FILLER                      PIC X(129).
```

```
      04 WSC-MSG-EXPIRY               PIC 9(9).
      04 WSC-MSG-PRIORITY             PIC 9(9).
      04 WSC-DATAGRAM-SYNCPOINT-CTL.
         06 WSC-SYNCPOINT-CTL-FLG     PIC 9.
            88 MQSYNC-APPL-CONTROLLED VALUE 1.
         06 WSC-MQCONN-HANDLE         PIC S9(9) BINARY.
      04 WSC-MQGET-DATA-CONVERSION    PIC 9.
         88 MQGET-CONVERT-DATA        VALUE 1.
02 WSC-OUTPUT-DATA.
   03 WSC-RETURN-CD              PIC S9(04) BINARY VALUE +0.
      88 NORMAL-COMPLETION            VALUE +00.
      88 ARRAY-OVERFLOW               VALUE +01.
      88 DATA-TRUNCATED               VALUE +02.
      88 OVERFLOW-AND-TRUNC           VALUE +03.
      88 INVALID-CALL                 VALUE -01.
      88 PROCESS-ERROR                VALUE -02.
      88 SERVICE-FAILURE              VALUE -03.
      88 SOAP-FAULT                   VALUE -04.
      88 CONNECTION-FAILURE           VALUE -05.
      88 DOM-ERROR                    VALUE -06.
      88 VALIDATION-FAILURE           VALUE -07.
      88 PROGRAM-ABEND                VALUE -99.
   03 WSC-RETURN-MSG             PIC X(100)     VALUE SPACE.
02 WSC-INVOKE-TRACE              PIC X(001)     VALUE 'N'.
02 WSC-WARNING-FLAG              PIC X(01)      VALUE 'I'.
   88 REPORT-WARNINGS VALUE 'R'.
   88 IGNORE-WARNINGS VALUE 'I'.
02 WSC-VALIDATE-SCHEMA           PIC X(01)      VALUE ' '.
   88 VALIDATE-REQ-SCHEMA  VALUE 'I' 'B'.
   88 VALIDATE-RESP-SCHEMA VALUE 'O' 'B'.
   88 VALIDATE-ALL-SCHEMA  VALUE 'B'.
02 WSC-FUTURE-USE                PIC X(002)     VALUE SPACE.
02 WSC-TIMEOUT-SECONDS           PIC S9(05) COMP-3 VALUE +0.
02 WSC-TIMEOUT-MICROSEC          PIC S9(05) COMP-3 VALUE +0.
02 WSC-CONNECTION-CLOSE          PIC X(01)      VALUE 'N'.
       88 CLOSE-CONNECTION                     VALUE 'Y'.
       88 REUSE-CONNECTION                     VALUE 'N'.
02 WSC-TCPIP-JOBNAME                PIC X(008) VALUE SPACE.
02 WSC-RELEASE-TMPL-STORAGE         PIC X(001) VALUE 'N'.
       88 ReleaseTemplateStorage      VALUE 'Y'.
02 FILLER                           PIC X(080) VALUE SPACE.
02 WSC-INTERNAL-USAGE               PIC X(118) VALUE SPACE.

02 WSC-METHOD-AREA                  PIC X(101500).
02 WSC-SOAP-Fault REDEFINES WSC-METHOD-AREA.
   10 WSC-Fault-Code        PIC X(50).
   10 WSC-Fault-String.
      15 WSC-Fault-Str-Len  PIC S9(04) BINARY.
      15 WSC-Fault-Str-Text PIC X(500).

02 Input-Request REDEFINES WSC-METHOD-AREA.
   03 TestLargeComplexData.
      04 LargeComplexDataIn.
         05 person
                                 OCCURS 500 TIMES.
            06 firstName         PIC  X(25).
            06 middileName       PIC  X(25).
```

```
                    06 lastName            PIC  X(25).
                    06 privacyId           PIC  S9(09) COMP.
********  The following comment refer to variable below
********  date format e.g - 2002-10-10+05:00
                    06 dateOfBirth         PIC  X(16).
                    06 phoneNumber         PIC  X(12)
                                           OCCURS 9 TIMES.
          02 Output-Response
            REDEFINES Input-Request.
             03 TestLargeComplexDataResponse.
                04 LargeComplexDataOut.
                   05 person
                                           OCCURS 500 TIMES.
                    06 firstName           PIC  X(25).
                    06 middileName         PIC  X(25).
                    06 lastName            PIC  X(25).
                    06 privacyId           PIC  S9(09) COMP.
********  The following comment refer to variable below
********  date format e.g - 2002-10-10+05:00
                    06 dateOfBirth         PIC  X(16).
                    06 phoneNumber         PIC  X(12)
                                           OCCURS 9 TIMES.
```

The major fields in the protocol section are described below:

WSC-ENVIRONMENT： Set this to 'O' for CICS programs, 'C' for containers, and 'B' for all other programs (IMS, Batch, DB2 Stored Procedure, etc). The default is 'O'.

WSC-METHOD-NAME: The web service operation name, extracted from the WSDL. Do not modify this field.

WSC-TEMPLATE-NAME: The name of the runtime metadata template. Do not modify this field.

WSC-TRANSPORT-PROT: Set this field to 'H' for http transport or M for MQ transport. The default is 'H.

WSC-SSL-IND: Set this field to 'Y' to use SSL security, 'A' for AT-TLS or 'N' for none. The default is 'N'. When the field is set to 'Y' then your outbound invocation will use native SOLA SSL support. This supports SSL 3.0 protocol and accepts server certificate having either 1024 or 2048-bit RSA keys. When the field is set to 'A' then your outbound invocation will exploit zOS TCPIP enabled AT-TLS (Application Transparent TLS).

Note: Contact your local zOS support to configure AT-TLS policy. AT-TLS supports SSL3.0 and TLS1.0 protocol.

http Data: The next few fields are only relevant to http transport.

WSC-NODE-1 thru WSC-NODE-4: Use these four fields to specify the 4 nodes of the IP address that your web service's binding endpoint. Leave these fields as zero if you want SOLA to use DNS to resolve your FQDN.

WSC-PORT: Use this field to specify the port number. By default this field is extracted from the port number of the soap:address location attribute.

WSC-FQDN: Use this field to specify the FQDN for your web service's binding endpoint. By default this field is extracted from the soap:address location attribute.

WSC-FILE-PATH: This field is extracted from the filepath of the soap:address location attribute.

WSC-PROXY: If your service is accessed through a proxy server then enter the FQDN of the proxy in this field.

WSC-PROXY-PORT: Specify the port number of the proxy in this field.

NOTE: SOLA Outbound Processor now supports passing extended file path URI by using the 122 byte storage of the WSC-PROXY field as additional storage. The application driver invoking the SOLA outbound processor has to set the WSC-PROXY-PORT field as follows:

```
MOVE 99999 TO     WSC-PROXY-PORT
```

and set the extended file path of up to 250 bytes by using storage areas of WSC-FILE-PATH (128 bytes) & WSC-PROXY (122 bytes).

```
04 WSC-FILE-PATH                PIC X(128)
   VALUE '/some/extended/filepath'.
04 WSC-PROXY                    PIC X(122) VALUE SPACE.
04 WSC-PROXY-PORT               PIC 9(05)  VALUE 0.
```

MQ Data: The next few fields are only relevant to MQ transport

WSC-CONVERS-TYPE: Specify the MQ conversation type in this field. 'D' for Datagram, 'R' for RequestReply or 'T' for ReplyTo.

WSC-MANAGER-NAME: The name of the MQ Queue Manager to connect to.

WSC-QUEUE-NAME: The name of the queue that SOLA should write to.

WSC-REPLY-TO-QUEUE: The name of the reply queue

WSC-MSG-EXPIRY: This represents the time (in milliseconds) that a message placed on a queue is allowed to persist before being removed by the queue manager. The default is to have the message persist indefinitely.

WSC-MSG-PRIORITY: The priority to be assigned to the message.

WSC-SYNCPOINT-CTL-FLG: The fields WSC-SYNCPOINT-CTL-FLG and WSC-MQHCONN-HANDLE go together. If the client wants to control sync/rollback operations when using outbound over MQ messaging protocol, then he must set WSC-SYNCPOINT-CTL-FLG. If this flag is set then he must also provide WSC-MQHCONN-HANDLE during the call which represents a particular MQ manager that SOLA will use during the outbound processing.

WSC-MQCONN-HANDLE: This represents a connection handle (automatically returned to the application after a request for a connection), that is, the connection to a particular queue manager. Normally when SOLA returns to an application during outbound calls, any MQ message processing has already automatically been "synced on return". If however the application wants to control sync/rollback operations itself, then the application must pass in a particular connection handle which SOLA will use during all outbound processing of MQ messages.

Please note if MQ is used: CSD Definition and XML9 definition for TRANCLASS is shipped with the default DFHTCL00 and must be customized at setup.

WSC-MQGET-DATA-CONVERSION: This tells SOLA whether to perform any data conversion on messages that are being retrieved on behalf of the application. The conversion of the message will be in accordance to the encoding used when the message was originally placed on the queue.

WSC-RETURN-CD: The return code issued by SOLA. Values are:

```
NORMAL-COMPLETION           VALUE +00
ARRAY-OVERFLOW              VALUE +01
DATA-TRUNCATED              VALUE +02
OVERFLOW-AND-TRUNC          VALUE +03
INVALID-CALL                VALUE -01
PROCESS-ERROR               VALUE -02
SERVICE-FAILURE             VALUE -03
SOAP-FAULT                  VALUE -04
CONNECTION-FAILURE          VALUE -05
DOM-ERROR                   VALUE -06
VALIDATION-FAILURE          VALUE -07
PROGRAM-ABEND               VALUE -99
```

WSC-RETURN-MSG: The error message issued by SOLA if WSC-RETURN-CODE is negative.

WSC-INVOKE-TRACE: Set this field to 'Y' to turn on a detailed trace. Use for debugging purposes only, as the volume of trace data can be large.

WSC-WARNING-FLAG: Set this field to 'R' to report validation failures, or 'I' to ignore validation failures. See the Validation section on page 104 for details.

WSC-VALIDATE-SCHEMA: Set this field to 'I' to validate requests, 'O' to validate responses or 'B' to validate both. See the Validation section on page 104 for details.

WSC-TIMEOUT-SECONDS and WSC-TIMEOUT-MICROSEC: How long SOLA should wait for a response from the remote web service before timing out.

WSC-CONNECTION-CLOSE: Whether SOLA should close the TCPIP connection when the service has completed. Specify 'Y' to close the connection or 'N' to leave the connection open for high-volume batch applications.

WSC-TCPIP-JOBNAME: The name of the TCPIP stack that SOLA should connect to. If this field is blank then SOLA will connect to 'TCPIP'.

WSC-RELEASE-TMPL-STORAGE: Flag to indicate if SOLA should release the loaded template when the service invocation is completed.
Note: Set this flag to '**Y**' if you are running under **IMS** and using SOLA Outbound plugin.

SOAP Faults

If a soap fault is returned by the remote service then WSC-RETURN-CD will contain -04 and the following fields will be populated with the text of the soap fault. The fault area redefines the soap input area.

WSC-Fault-Code: A Code that represents the fault
WSC-Fault-String: A string containing the text of the soap fault.

SOAP Request Area

This area will contain the fields that the remote service requires you to provide. In the example of the nameSearch service, the fields are:

```
02 Input-Request REDEFINES WSC-METHOD-AREA.
   03 nameSearch.
      04 BossID                    PIC  X(8).
      04 SearchValue               PIC  X(25).
```

SOAP Response Area

This area will contain the fields that the remote service returns. In the example of the nameSearch service, the fields are:

```
02 Output-Response
  REDEFINES Input-Request.
   03 nameSearchResponse.
      04 Dfhcommarea.
         05 ReturnCode             PIC  S9(04) COMP.
         05 ReturnMsg              PIC  X(100).
         05 SequelCode             PIC  S9(04) COMP.
         05 CICSReturnCode         PIC  X(4).
         05 HostSysid              PIC  X(4).
         05 TotalCounter           PIC  S9(04) COMP.
         05 FetchCounter           PIC  S9(04) COMP.
         05 ClientInfo
                                  OCCURS 300 TIMES.
            06 ClientName          PIC  X(45).
            06 ProducerID          PIC  X(7).
            06 ClientNumber        PIC  S9(09) COMP.
            06 PhoneNumber         PIC  X(20).
```

Validation

SOLA has the capability to validate runtime data according to the schema data type. There are two protocol fields in the copybook that are used for validation: WSC-VALIDATE-SCHEMA and WSC-WARNING-FLAG.

To enable validation, WSC-VALIDATE-SCHEMA must be set to 'I' (to validate requests), 'O' (to validate responses) or 'B' (to validate both).

Additionally, you can instruct SOLA to report or to ignore validation failures. Setting WSC-WARNING-FLAG to 'R' will report validation failures, while setting it 'I' will ignore them.

WSC-RETURN-CD will return a code of -07 if WSC-WARNING-FLAG is set to 'R'.

Using SOLA to Invoke Outbound Requests

At runtime, it's fairly simple to invoke an Outbound web service. All that's required is to fill in the input fields in the generated copybook, override any fields in the protocol section, and call the SOLA Outbound utility module XMLPC103.

Because the values in the protocol section were extracted from your WSDL, you may need to override them. For example, when you imported the WSDL the soap:address may have referenced a version of the outbound service that resides on a development server, but in your test and production systems you would want to use a test or production version of the service. This can be accomplished using virtual services defined in SOA Software's Service manager, or by overriding the FQDN field in the WSC-XMLPC103-LINKAGE copybook.

Invoking an outbound service from CICS

Format 1. Copybook less than 32k

```
01  WS-XMLPC103                PIC  X(08)   VALUE 'XMLPC103'.

EXEC CICS LINK
   PROGRAM (WS-XMLPC103)
   COMMAREA(WSC-XMLPC103-LINKAGE )
   LENGTH  (LENGTH OF WSC-XMLPC103-LINKAGE)
   RESP    (WS-RESP)
   RESP2   (WS-RESP2)
END-EXEC
```

Format 2. Copybook greater than 32k

First place the SOLA generated copybook in a container called 'SOLA-CONTAINER' (the Channel and Container names are important and must be as shown).

```
MOVE 'SOLA-CHANNEL'                  TO WS-SOLA-CHANNEL
MOVE 'SOLA-CONTAINER'                TO WS-SOLA-CONTAINER
MOVE LENGTH OF WSC-XMLPC103-LINKAGE TO WS-CONTAINER-LEN

EXEC CICS PUT
   CONTAINER(WS-SOLA-CONTAINER)
   Channel   (WS-SOLA-CHANNEL)
   From      (WSC-XMLPC103-LINKAGE)
   FLENGTH   (WS-CONTAINER-LEN)
   RESP      (WS-RESP)
   RESP2     (WS-RESP2)
END-EXEC
```

Now link to XMLPC103 passing the Channel.

```
EXEC CICS LINK
   PROGRAM (WS-XMLPC103)
   Channel (WS-SOLA-CHANNEL)
   RESP    (WS-RESP)
   RESP2   (WS-RESP2)
END-EXEC
```

Invoking an outbound service from Batch, IMS, DB2 Stored Proc, etc.

```
01  WS-XMLPC103                PIC  X(08)   VALUE 'XMLPC103'.

CALL WS-XMLPC103 USING WSC-XMLPC103-LINKAGE
```

Using WS-Security with Outbound requests

The current version of SOLA doesn't support WS-Policy for Outbound web services requests. SOLA does however expose its internal Policy interface to allow web service requestors to influence the creation of the wsse:Security header on Outbound requests.

This interface is remarkably simple to use; all that's required is to provide a second copybook containing the security credentials. The methods of calling SOLA with the second copybook are limited to:

- CALL USING WSC-XMLPC103-LINKAGE, WSC-SECURITY-TOKEN for non CICS calls
- EXEC CICS LINK using two containers, the first containing WSC-XMLPC103-LINKAGE and the second containing the WSC-SECURITY-TOKEN.

The Security Structure WSC-SECURITY-TOKEN is provided in the SOLA SAMPLIB as member SECTOKEN. Include this member in your program and populate it before calling XMLPC103.

```
 01  WSC-SECURITY-TOKEN.
     05  WSC-Eye-Catcher             PIC  X(08) VALUE 'SECTOKEN'.
     05  WSC-Security-Head-Cnt       PIC S9(04) VALUE +0 BINARY.
     05  WSC-Security-Info OCCURS 3 TIMES.
*-------------------------------------------------
*-------must understand is attribute of the security
*-------if you chose to ignore it, fill it with spaces.
*-------------------------------------------------
         10  WSC-Must-Understand     PIC  X(01).
             88  MustUnderStand-0    VALUE '0'.
             88  MustUnderStand-1    VALUE '1'.
             88  OmitMustUndrStnd    VALUE ' '.
         10  Actor                   PIC X(64).
         10  WSC-Add-Timestamp         PIC  X(01).
             88  No-Timestamp          VALUE 'N' ' '.
             88  Add-Timestamp         VALUE 'T'.
         10  WSC-Token-Timestamp.
*-------------------------------------------------
*---- Create must be sent in the following format
*----  example - 2009-11-12T05:13:51.428Z
*---- Fill create with spaces if not provided.
*---- If spaces, SOLA will use current GMT timestamp
*-------------------------------------------------
             15  T-Create-Token-Tm     PIC  X(26).
*-------------------------------------------------
*-------------------------------------------------
*--- Expire may be sent in the format 2009-11-12T05:13:51.428Z
*--- Fille it with spaces or low value if not provided
*-------------------------------------------------
             15  T-Expire-Token-Tm     PIC X(26).
*-------------------------------------------------
*--- Expire may also be given as interval from current time
*--- in seconds
*-------------------------------------------------
             15  T-Expire-Interval.
                 20  T-expire-Seconds  PIC S9(09) BINARY.
         10  FILLER                  PIC X(256).
         10  WSC-Token-Cnt           PIC S9(04) BINARY.
         10  WSC-Token-Data OCCURS 3 TIMES.
             15  WSC-Token-Type      PIC  X(01).
                 88  UserNameToken   VALUE 'U'.
                 88  Custom-Token    VALUE 'C'.
*-------------------------------------------------
*---------------Provide the user name --------
*-------------------------------------------------
             15  Username-Token      PIC  X(256).
*-------------------------------------------------
*--------If custom token is sent, you need to ----
*--------pass the entire xml text ----------------
*-------------------------------------------------
             15  Custom-Token-Data   REDEFINES
                 Username-Token.
                 20  Cust-Token-Len  PIC S9(09) COMP.
                 20  Cust-Token-Ptr  POINTER.
                 20  FILLER          PIC  X(248).
*-------------------------------------------------
*---------If password is not sent in the clear text
```

```
*---------a digest can be passed
*-------digest password = BASE64(SHA-1(Nonce+create+password))
*-------------------------------------------------
            15  Password-type-ind   PIC  X(01).
                88  Clear-text      VALUE 'C' ' '.
                88  Digest-b64      VALUE 'B'.
*-------------------------------------------------
*-If your password is in cleare text or digest format
*-set d-no-action to true (when sending digest, use base64 format)
*-If you want sola to generate digest then
*-Set Generate-digest to true (ICSF must be active)
*-In this case SOLA will generate digest as follows
*-digest password = BASE64(SHA-1(Nonce+create+password))
*-SOLA will generate digst from password after converting to utf8
*-------------------------------------------------
            15  Password-action     PIC  X(01).
                88  d-no-action       VALUE 'N' ' '.
                88  Generate-digest VALUE 'G'.
            15  Password-Token      PIC  X(128).
*-------------------------------------------------
*-----------Below is example of nonce format
*---------- If clear-text and no-action
*----------     fill nonce with spaces
*----------     must fill password with clear text ebcdic
*---------- If digest-b64 and generate-digest
*----------     Either fill nonce with binary
*----------     or fill it with space so SOLA will generate
*----------     fill password with clear text (ebcdic)
*----------     fill create with value or space for sola to
*----------     generate
*---------- If digest-b64 and no-action
*----------     must fill nonce with base64 value
*----------     must fill password with base64 digest
*----------     must fill create
*---------- If clear-text and generate-digest
*----------     generate-digest is ignored
*-------------------------------------------------
            15  Password-Nonce      PIC  X(128).
*-------------------------------------------------
*--------Nonce encoding attribute is for future use
*--------If encoding type is set to spaces,
*--------this attribute will not be populated
*-------------------------------------------------
            15  Nonce-encoding      PIC  X(1).
                88  base64-binary   VALUE 'B' ' '.
*-------------------------------------------------
*---- Create must be sent in the following format
*----  example - 2009-11-12T05:13:51.428Z
*---- Fill create with spaces if not provided.
*---- If spaces, SOLA will use current GMT timestamp
*-------------------------------------------------
            15  Create-Token-Tm     PIC  X(26).
*-------------------------------------------------
*-------The timestamp token will be filled
*-------If you indicate add-timestamp
*-------Timestamp element is added to Security header
*-------outside username token if present
```

```
*-------------------------------------------------
*-------------------------------------------------------
*--- WS-Addressing for future use only ----------------
*-------------------------------------------------------
         10  WSC-Addressing.
             15  Addressing-Container.
                20  Add-Value-Len          PIC S9(09) COMP.
                20  Add-Value-Ptr          Pointer.
                20  Add-Value-filler       PIC X(08).
*-------------------------------------------------------
*-----------------For future only-----------------------
*-------------------------------------------------------
     05  WSC-RM.
         10  RM-Container.
             15  RM-Value-Len          PIC S9(09) COMP.
             15  RM-Value-Ptr          Pointer.
             15  RM-Value-filler       PIC X(08).
*-------------------------------------------------------
*-----------------For future only-----------------------
*-------------------------------------------------------
     05  WSC-XML-Encryption.
         10  Enc-Container.
             15  Enc-Value-Len          PIC S9(09) COMP.
             15  Enc-Value-Ptr          Pointer.
             15  Enc-Value-filler       PIC X(08).
*-------------------------------------------------------
*-----------------For future only-----------------------
*-------------------------------------------------------
     05  WSC-XML-Signature.
         10  Sig-Container.
             15  Sig-Value-Len          PIC S9(09) COMP.
             15  Sig-Value-Ptr          Pointer.
             15  Sig-Value-filler       PIC X(08).
     05  Filler                      PIC  X(256).
     05  WSC-Http-Head-Cnt           PIC S9(04) VALUE +0 BINARY.
*-------------------------------------------------
*------------Repeat the header info based on count-
*-------------------------------------------------
     05  WSC-HTTP-Header-Info OCCURS 1 TO 15 TIMES
                              DEPENDING ON WSC-Http-Head-Cnt.
*-------------------------------------------------
*-----------Header value can be given--------------
*-----------as text up to 128 bytes  --------------
*-----------or len + pointer in case of batch program
*-----------or CICS Container name that contains the value
*-------------------------------------------------
         10  Http-Value-Ind          PIC  X(1).
                 88  Value-given     VALUE 'V' ' '.
                 88  Pointer-given   VALUE 'P'.
                 88  Container-given VALUE 'C'.
*-------------------------------------------------
*------Below is an example of name value pair -----
*-------------------------------------------------
*------Cookie: $Version=1; UserId=JohnDoe
*------Accept: */*
*------If-None-Match: "737060cd8c284d8af7ad3082f209582d"
*-------------------------------------------------
```

```
*------You can also put custom headers such as------
*------custom-header: <some value >
*-------------------------------------------------
         10  Http-Name-value-pair    PIC  X(256).
*-------------------------------------------------
*----------- Fill container name if using CICS Containers
*----------- else use pointer if the value is bigger than 256
*-------------------------------------------------
         10  Http-Nm-value-Long.
             15  Http-Nm-Value-Len       PIC S9(09) COMP.
             15  Http-Nm-Value-Container.
                 20  Http-Nm-Value-Ptr       Pointer.
                 20  Http-Nm-Value-filler    PIC X(12).
*****05  WSC-Additional-HTTP-Header  PIC S9(04) value zero.
     05  Filler                      PIC  X(256).
*-------------------------------------------------
*  if additional header needs to be added on http
*  please uncomment 3 lines below and keep repeating
*  them for each additional header
*-------------------------------------------------
*  05 WSC-NO-OF-HEADERS.
*      10  WSC-Header        PIC X(128).
*      10  WSC-Header-value-len PIC S9(04).
*      10  WSC-Header-value  PIC X(256).
***-----------------------------------------------------
***-- the above WSC-Header-value can be expanded up to 4k
***-- please make sure WSC-HEADER-value-len contains the
***-- corrent length else wrong data will be sent
***-----------------------------------------------------
***-- End of security section
***-----------------------------------------------------
*-------------------------------------------------
*------Encryption and signatures are for future use only
*------------ Not currently supported --------------
*-------------------------------------------------
 01  WSC-Encryption-Decryption.
     05  WSC-Encryption.
*--Future only---------RSA Key must be defined in ICSF
         10 WSC-Enc-ICSFKey         PIC X(64).
*--Future only---------Cert Container or pointer
         10 WSC-Enc-Cert-Container.
            15 WSC-Enc-Cert-Pointer  usage is pointer.
            15 WSC-Enc-Cert-FIller   PIC X(12).
*--Future only--------can define partial x path to fit
*---------------------- in 256 bytes
         10 WSC-Enc-Element          PIC X(256) occurs 3 times.
     05  WSC-Decryption.
*--Future only---------RSA Key must be defined in ICSF
         10 WSC-Dec-ICSFKey         PIC X(64).
 01  WSC-Signature.
     05  WSC-Signature-create.
*--Future only---------RSA Key must be defined in ICSF
         10 WSC-Sign-ICSFKey         PIC X(64).
*--Future only--------can define partial x path to fit
*---------------------- in 256 bytes
         10 WSC-Sign-Element          PIC X(256) occurs 3 times.
     05  WSC-Sign-Verify.
```

```
*--Future only---------RSA Key must be defined in ICSF
         10 WSC-Veri-ICSFKey          PIC X(64).
         10 WSC-Veri-Cert-Container.
            15 WSC-Veri-Cert-Pointer  usage is pointer.
            15 WSC-Veri-Cert-FIller   PIC X(12).
```

Here's a brief description of how to use the fields in the SECTOKEN copybook:

WSC-Security-Head-Cnt: Indicates the number of Security Headers to include.

WSC-Must-Understand: A one byte field. Values '0' '1' and ' '. A blank causes the attribute to be omitted entirely.

Actor: If value is other than spaces then it will be added as the 'actor' attribute to the wsse:Security element.

WSC-Token-Cnt: Indicates the number of user name tokens (wsse:UsernameToken) to be added (up to 3).

WSC-Token-Type: 'C' for custom (not yet supported) or 'U' for Username Token (supported)

Username-Token: The user name token.

Password-type-ind: 'C' ' ' or 'B'. This indicates if the password is clear text ('C' or blank) or a base64 digest ('B')

Password-action: 'N', ' ' or 'G'. 'N' or blank indicates that there is no action needed on the Password. Either you should provide (and wish to use) plain text, or you must provide the base64 digest yourself. 'G' indicates to SOLA that we need to generate the digest using the provide plain text password.

Password-Token: The clear text password of the base64 digest password.

Password-Nonce: If digest is used this is the value of the Nonce.

Nonce-encoding: For future use. Currently only base64 is supported.

Create-Token-Tm: If using a digest this is the Created time

WSC-Add-Timestamp: 'T' if you want SOLA to add a timestamp to your request.

Invoking an Outbound Service from CICS using WS-Security

Please refer to program WCC6032A, which is shipped in the SOLA SAMPLIB, for a sample program that invokes the WS-Security interface from a CICS program.

An abbreviated description of the process is as follows:

First place the SOLA generated copybook in a container called 'SOLA-CONTAINER' (the Channel and Container names are important and must be as shown).

```
01  WS-XMLPC103                PIC  X(08)   VALUE 'XMLPC103'.

MOVE 'SOLA-CHANNEL'                 TO WS-SOLA-CHANNEL
MOVE 'SOLA-CONTAINER'               TO WS-SOLA-CONTAINER
MOVE LENGTH OF WSC-XMLPC103-LINKAGE TO WS-CONTAINER-LEN

EXEC CICS PUT
   CONTAINER(WS-SOLA-CONTAINER)
   Channel  (WS-SOLA-CHANNEL)
   From     (WSC-XMLPC103-LINKAGE)
   FLENGTH  (WS-CONTAINER-LEN)
   RESP     (WS-RESP)
   RESP2    (WS-RESP2)
END-EXEC
```

Then place the security token data structure into a container called 'SOLA-SECURITY' as shown.

```
MOVE 'SOLA-CHANNEL'                 TO WS-SOLA-CHANNEL
MOVE 'SOLA-SECURITY'                TO WS-SOLA-CONTAINER
MOVE LENGTH OF WSC-SECURITY-TOKEN TO WS-CONTAINER-LEN

EXEC CICS PUT
   CONTAINER(WS-SECUR-CONTAINER)
   Channel  (WS-SOLA-CHANNEL)
   From     (WSC-SECURITY-TOKEN)
   FLENGTH  (WS-CONTAINER-LEN)
   RESP     (WS-RESP)
   RESP2    (WS-RESP2)
END-EXEC
```

Now link to XMLPC103 passing the Channel.

```
EXEC CICS LINK
   PROGRAM (WS-XMLPC103)
   Channel (WS-SOLA-CHANNEL)
   RESP    (WS-RESP)
   RESP2   (WS-RESP2)
END-EXEC
```

Invoking an Outbound Service from Batch, IMS, DB2 Stored Proc, etc using WS-Security

Please refer to program WCC6032B, which is shipped in the SOLA SAMPLIB, for a sample program that invokes the WS-Security interface from a non-CICS program.

An abbreviated description of the process is as follows:

Populate the two copybooks then call the SOLA Outbound utility module XMLPC103. The example below shows a COBOL version of the call.

```
01  WS-XMLPC103                PIC  X(08)   VALUE 'XMLPC103'.
```

```
CALL WS-XMLPC103 USING WSC-XMLPC103-LINKAGE
                     , WSC-SECURITY-TOKEN
```

Analyzer Reference

This section contains information about the various menu options, properties and alternate views available in the analyzer. You can use this reference to increase your familiarity with the analyzer, as well as learn how to perform more complex analysis using the full capabilities of SOLA Developer.

Analyzer Button Bar

Prefix Exclusion Field: this field allows for a prefix or a group of prefixes to be entered for exclusion from the field names. If more than one prefix is entered, prefixes must be separated by commas. The prefix field works as a preprocessing step when **APPLY DICTIONARY** is clicked. The unwanted prefixes are first stripped off, and the resulting names are then fed into the dictionary.

APPLY DICTIONARY: this button applies the SOLA dictionary to every item in either the legacy tree or the schema tree, depending on what type of analysis you are doing (outbound, inbound, etc.). For more information about the SOLA dictionary, see page 246.

FINALIZE: this button will finalize the analysis, and create the method.

Legacy and Schema Trees

Depending on the type of web service you are creating (bottom up, top down, etc.), which one of the two tree types (legacy or schema) is the target of your analysis may differ. However, the drag and drop functionality is identical regardless of analysis type.

Tree Placement

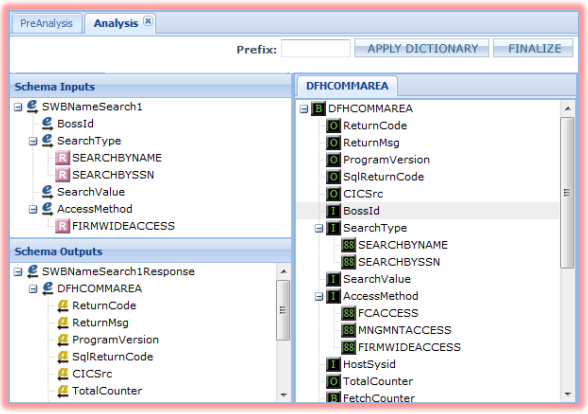
The location of the trees will vary depending on which type of analysis you are doing.

Inbound, bottom up analysis: the legacy tree is on the left, and the schema tree, which is the target of the analysis (the one you build and/or configure when analyzing) is on the right. This includes callable and channel/container analysis, which is always inbound bottom-up.

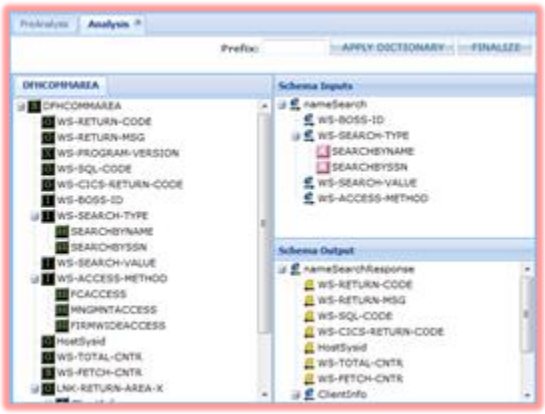
Outbound or inbound top-down analysis: the schema tree is on the left, and the legacy tree, which is the target of the analysis (the one you build and/or configure when analyzing) is on the right.

Inbound, meet-in-the-middle analysis: the legacy tree is on the left, and the schema tree is on the right. Both trees are considered as source trees and so are frozen from any drag and drop updates. The only user update permitted is the establishing of links between legacy and schema tree elements.

When dragging items from one tree to another, you drag from the left tree to the right tree.



Outbound & Inbound Top Down



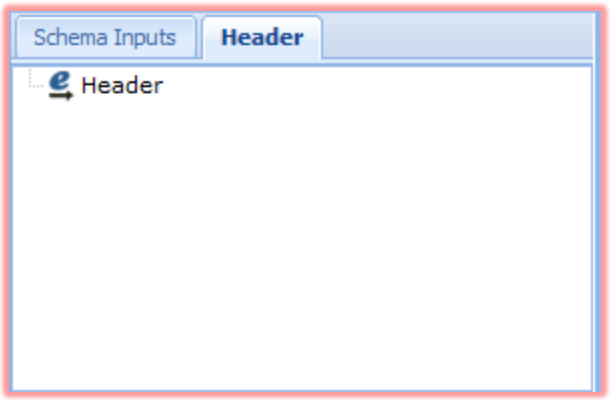
All other types

The tree on the left is referred to as the "source tree", because that's the structure you will use to build either a WSDL (inbound bottom-up, callable, container) or a copybook (outbound, inbound top-down).

The tree on the right is called the "target tree", because that's the tree you are building to create your web service.

Header Tabs

Both the Schema Inputs and Schema Outputs section have a Header tab. This allows you to configure the input and output SOAP header for the web service you are creating.



Adding elements to the headers works the same way as adding elements to the input our output sections of the schema.

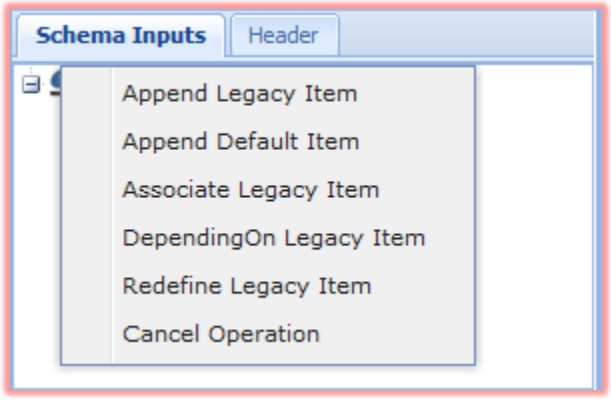
Click on the header tab (input or output) you want to configure, and drag items into that section.

Drag and Drop Operations

The legacy and schema trees support the following drag and drop operations:

Copy item from one tree to another: you can drag and drop items from the source tree (left) to the target tree (right). When building a web service, depending on the type of analysis (inbound, outbound, etc.), you are either building a WSDL or a copybook. Dragging items from the source tree to the target tree is how that WSDL or copybook is constructed. Detailed information about how to conduct an analysis begins on page 45. When you drag an item from the source tree to the target tree, the item remains in the source tree and an equivalent item ends up in the target tree. The destination item will be linked (associated) with the source item (see below).

CTRL Copy item from one tree to another: you can access a special menu of drag and drop operations by holding down the CTRL key when you drag an item from one tree to another.



- **Append Legacy Item**: this is the default drag and drop operation and is the same as not using the CTRL key.
- **Append Default Item**: this moves the item from the left tree to the right tree, but sets its node type as "default". This means that it will not be in the schema and SOLA will pass a default value to the legacy program. You will need to set the value using the properties panel.
- **Associate Legacy Item**: when an item is moved from the source tree (left) to the target tree (right), it will have an association with its source. That way, you will always know the source of the item in the tree you are building and/or configuring,

even if you change its name. However, if you create a new item in the target tree, it will not have an associated source tree item. You can then use this CTRL-drag operation to create an association for that item. You can also use this operation to override an existing association and create a new one.

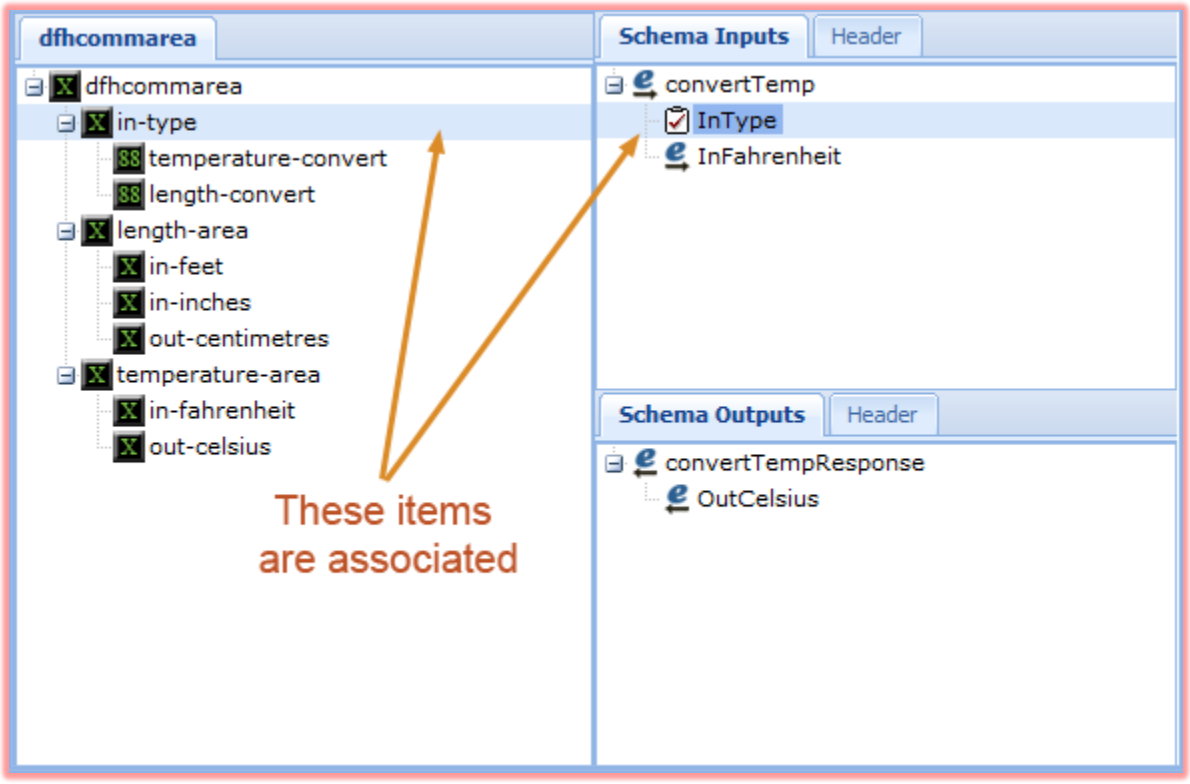
- **DependingOn Legacy Item**: you can use this operation to create an occurs-depending-on link from an item from the target tree to an array in the source tree. The target array's number of occurrences will then be limited to the numerical value of the linked item. For example, if there is a value in the source tree called "fetchCounter" with a default value of 100 and you drag fetchCounter on top of an array in the target tree, that array will be limited to 100 occurrences. This is for inbound bottom-up analysis only (including callable and channel/container).

- **Redefine Legacy Item**: this operation is only used when the legacy tree is the source tree (bottom up and meet-in-the-middle) and creates a different kind of association. When you use this operation, the legacy item you drag to the schema tree will be redefined by the legacy item the target schema item is associated with. You can use this to control memory usage by the legacy program.

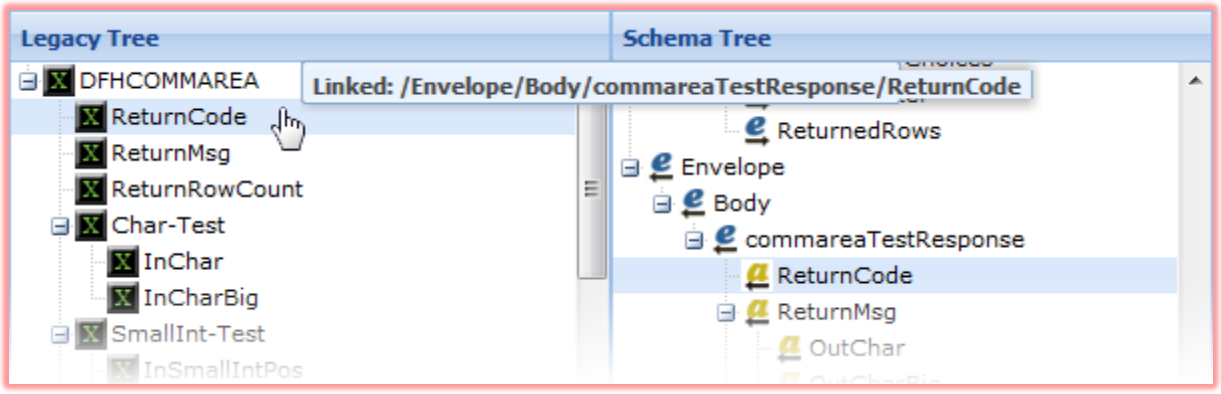
- **Cancel Operation**: cancels the drag and drop operation.

Display relationship between an item in one tree and its counterpart in another tree: when an item is moved from the source tree (left) to the target tree (right), it will have a permanent association with its source. That way, you will always know the source of the item in the tree you are building and/or configuring, even if you change its name.

This relationship is displayed via a highlight. When you click on an item in the target tree, its corresponding item in the source tree will be highlighted.



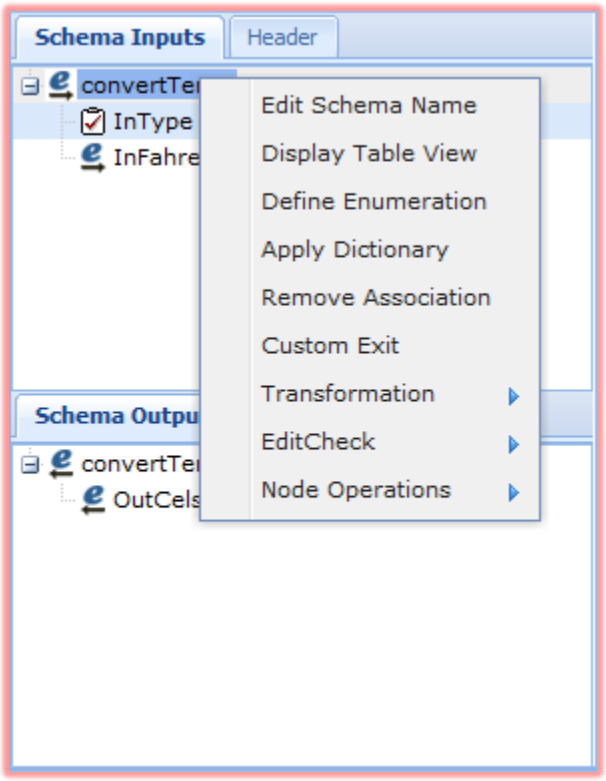
This association is also displayed in a pop-up dialog when you hover over a tree item on either tree.



Move item within the same tree: you can move items around in the right tree to change their position in the WSDL or copybook. Moving items can have effects on functionality.

Tree Item Menus

Right-clicking on an item in either the legacy tree or the schema tree will display a pop-up menu. The menus are different, depending on whether you've clicked on a Legacy tree or a Schema tree. The menus contain several options for analyzing the program. Depending on which tree you chose and what type of analysis you are doing (inbound bottom-up, inbound top-down or outbound), you will see different items in the menu.



The complete list, for both the legacy and the schema trees, is presented here.

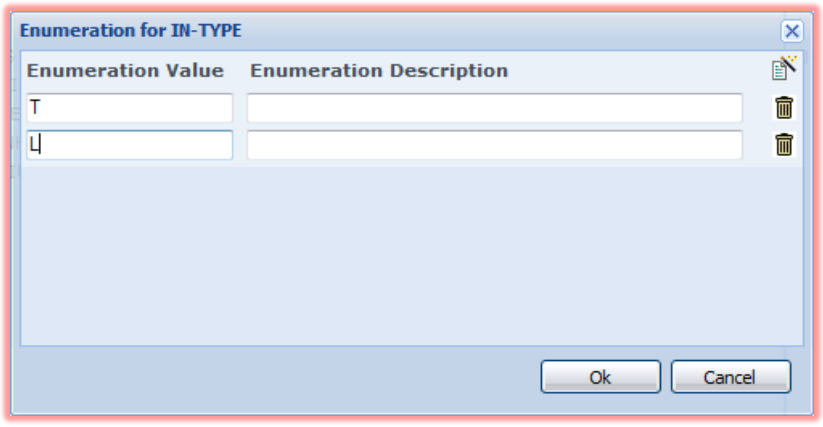
Apply Dictionary: choosing this option will apply the SOLA dictionary to the selected item only. For more information about the SOLA dictionary, see page 246.

Assign Default: Applies only to Default nodes. Allows user to attach a value in cases where this tag is not sent up as part of the SOAP Request.

Assign Container: Specify a container name to associate with a Legacy 01 level item.

Custom Exit: (Optional). Specify the name of an exit program to be called to perform custom transformations. Specifications on writing a custom exit will be provided on request.

Define Enumeration: choosing this option will display the enumerations panel, which lets you make changes to existing enumerations, create new enumerations or delete enumerations. The enumeration panel contains all the existing enumerations of the item you clicked on. If there are no enumerations, the panel will be blank.

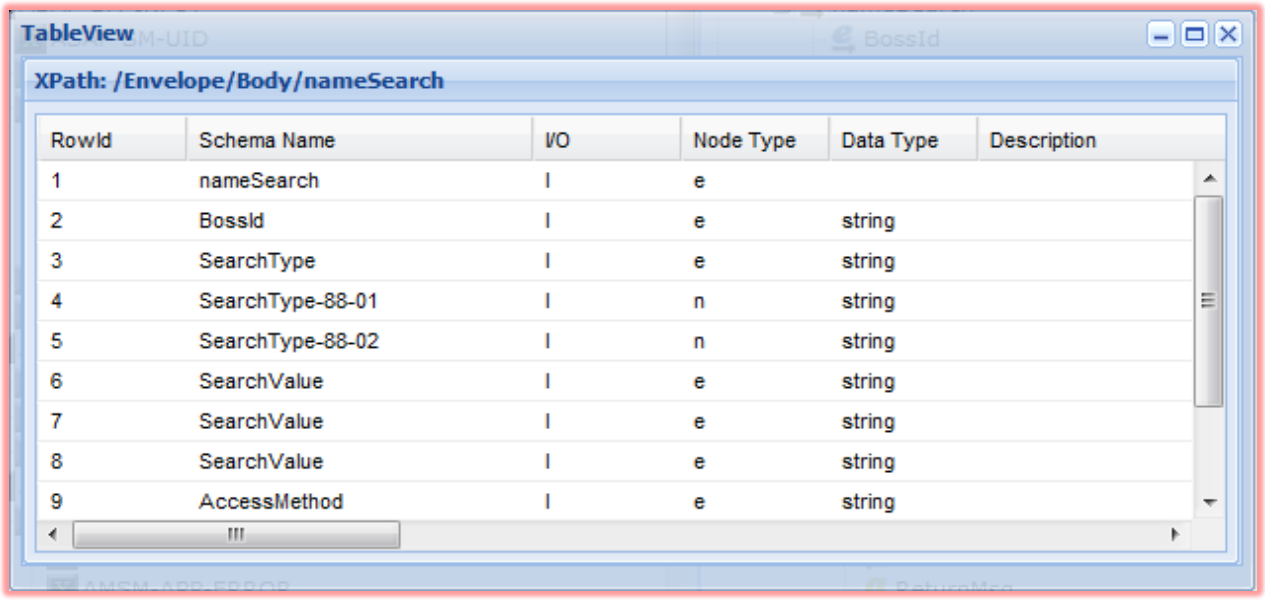


You can create new enumerations by clicking the icon and delete existing enumerations by clicking the icon next to the enumeration you wish to delete. To delete the L enumeration, click on its associated icon.

To change the value of an enumeration, enter the new value in the field under the **Enumeration Value** column. You can also add an optional description under the **Enumeration Description** column.

When you are finished with the enumerations panel, click Ok to save your changes or Cancel to discard them.

Display Table View: choosing this option will display the current item and all of its children in a table view (users of SOLA 5.x will recognize this familiar layout). The table view shows all of the items (parent and children) in a table under a series of column headings that correspond to the item's properties (from the properties panel).



Some users find it more convenient to change the various properties of an item and its children using the table view. There is nothing that you can do in a table view that you cannot do by selecting individual items and changing their values using the properties panel; the table view is offered as a time saving convenience to those users who prefer this type of layout.

Delete this Node: choosing this option deletes the selected item.

Edit Check: (Optional). Specify the name of an edit check program to be called to perform field validation. Specifications on writing an edit check program will be provided on request.

Edit Schema Name: The element or attribute name that appears in the schema may be modified.

Edit Legacy Name: use this to change the name of the legacy field. You can also change the name by double clicking the field.

Init Character: Specify an initialization character for a Legacy 01 level item. This is a single character that will be used to initialize the structure by copying that character to every byte in the structure. **Init Character** can only be specified at the 01 Structure name field, specifying it

for any other field in the structure will have no effect. Init Character can be specified on one of two ways:

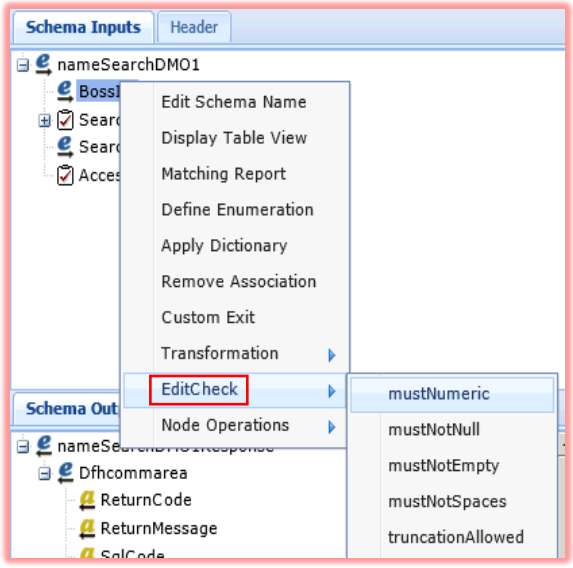
1. As a single displayable character
2. As a hexadecimal character, in the format X'00', which represents the a low-values character.

Input Processing: for input elements only (applies to Outbound analysis). Displays the input processing sub-menu, detailed below:

- **ExcludeIf:** choosing this option displays a sub-menu of additional options. Selecting an option from the sub-menu will exclude the selected field from the WSDL if certain conditions are met. Options are:
  - Default: will exclude the field if it's equal to its default value (spaces for character fields,zero for numeric fields).
  - Zero: will exclude the field if its value is 0 (zero).
  - Spaces: will exclude the field if its value is one or more spaces.
  - Low Value: will exclude the field if its value is binary zero.
  - HighValue: will exclude the field its value is all hexadecimal FF.

- **StopArrayIf:** choosing this option displays a sub-menu of additional options. Selecting an option from the sub-menu gives you the ability to pick an elementary item within an array to use as a sentinel to stop table processing if certain conditions are met (based on selected option)
  - Default: will stop the array if it's equal to its default value (spaces for character fields,zero for numeric fields).
  - Zero: will stop the array if the field's value is 0 (zero).
  - Spaces: will stop the array if the field's value is one or more spaces.
  - Low Value: will stop the array if its value is binary zero.
  - HighValue: will stop the array if its value is all hexadecimal FF.

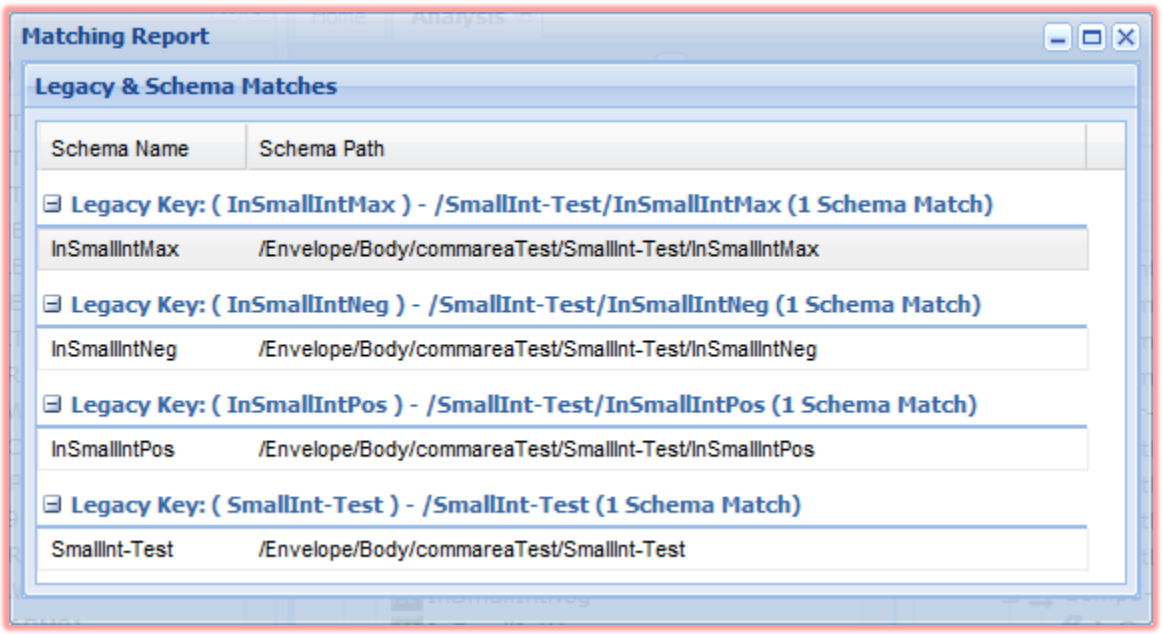
EditChecks on Input Processing: for input elements only (applied to Inbound analysis). You can set edit checks on input elements to do basic checks like mustNumeric, mustNotNull, mustNotEmpty, mustNotSpaces. SOLA will validate and reject the request if any of these checks, if specified on a field, fail.



SOLA by default validates lengths of the fields to ensure requests are rejected with 'Schema violation error' if data passed in any element is more than length of the program structure variable. If you desire to ignore these schema violations and allow for data to be truncated and passed to application then you can set 'truncationAllowed' edit check on individual input elements or you can set it at a more global level

on the first node (method node) in the Schema Inputs tree. Setting 'truncationAllowed' on the first node ensures that schema violations due to more data being passed in all the input elements of the method are tolerated.

Matching Report: this option is available only in the legacy tree. It displays the 'Legacy & Schema Matches dialog', which displays all the associations (links) between the selected legacy item (citem) and its children, and items in the schema tree (sitems). Clicking on the top level items will display a report of all associations in the tree.



Node Operations: displays the node operations sub-menu, detailed below:

- **All Attrs -> Elems:** changes the selected item and all of its child nodes from an attribute to an element.
- **All Elems -> Attrs:** changes the selected item and all of its child nodes from an element to an attribute.
- **Appent Child Node:** choosing this option will create an item (Citem or Sitem, depending on which type of analysis you are doing) and append it to the selected item as a child. The newly created item will be named "DoubeClick to Edit", indicating that you should name the node by double clicking its placeholder name.
- **Current Node -> Attr:** changes the current node (but not its child nodes) into an attribute.
- **Current Node -> Default:** changes the current node (but not its child nodes) into a default node.
- **Current Node -> Elem:** changes the current node (but not its child nodes) into an element.
- **Current Node -> Text:** changes the current node (but not its child nodes) into a text node.
- **Delete this Node:** choosing this option deletes the selected item.

- **Insert Node Before:** choosing this option will create an item and place it in the tree before the selected item on an equal level (i.e. if the selected item is a child node, the newly created item will also be a child of the same parent).

Output Processing: for output elements only (applies to Inbound analysis). Displays the output processing sub-menu, detailed below:

- **ExcludeIf:** choosing this option displays a sub-menu of additional options. Selecting an option from the sub-menu will exclude the selected field from the WSDL if certain conditions are met. Options are:
  - Default: will exclude the field if it's equal to its default value (spaces for character fields,zero for numeric fields).
  - Zero: will exclude the field if its value is 0 (zero).
  - Spaces: will exclude the field if its value is one or more spaces.
  - Low Value: will exclude the field if its value is binary zero.
  - HighValue: will exclude the field its value is all hexadecimal FF.

- **StopArrayIf:** choosing this option displays a sub-menu of additional options. Selecting an option from the sub-menu gives you the ability to pick an elementary item within an array to use as a sentinel to stop table processing if certain conditions are met (based on selected option)
  - Default: will stop the array if it's equal to its default value (spaces for character fields,zero for numeric fields).
  - Zero: will stop the array if the field's value is 0 (zero).
  - Spaces: will stop the array if the field's value is one or more spaces.
  - Low Value: will stop the array if its value is binary zero.
  - HighValue: will stop the array if its value is all hexadecimal FF.
  - NoChildren: For multi-level Group arrays – when the first level does not contain any fields to trigger stop array processing, use NoChildren. This will generate a property processing code '3105' to tell runtime while building the SOAP response to check group array elements and if there are no child nodes then the empty group node is deleted and Stop-array condition is triggered. See below examples.

Note: in Figure 1 below the first level array without elementary data items is set to StopArrayIf 'NoChildren' and in Figure 2 the property panel contains the new Processing Code '3105'.

Figure 1:

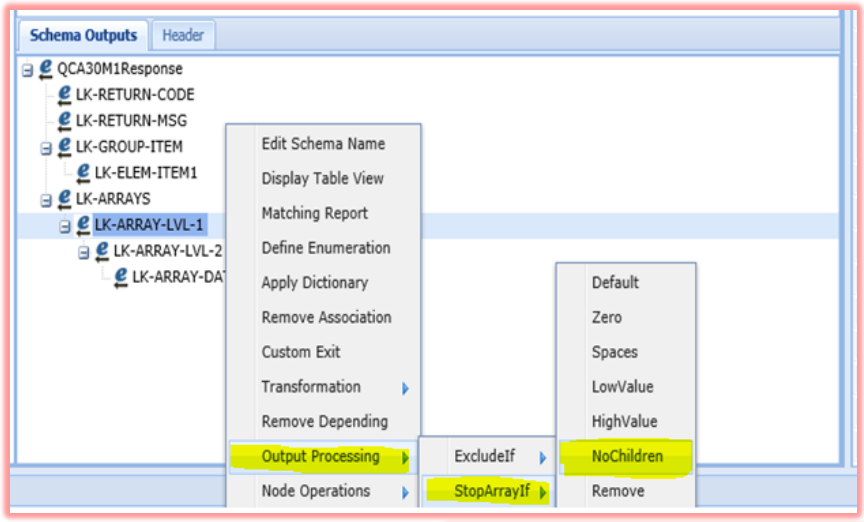
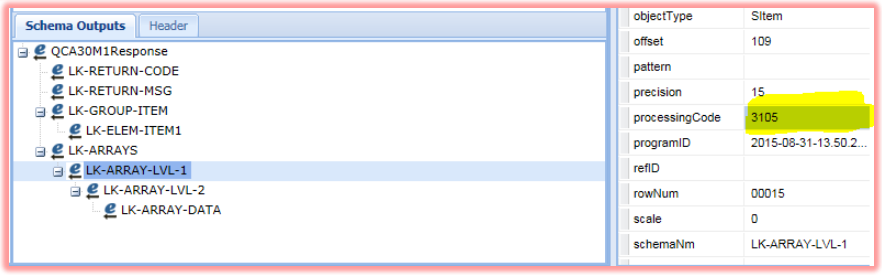
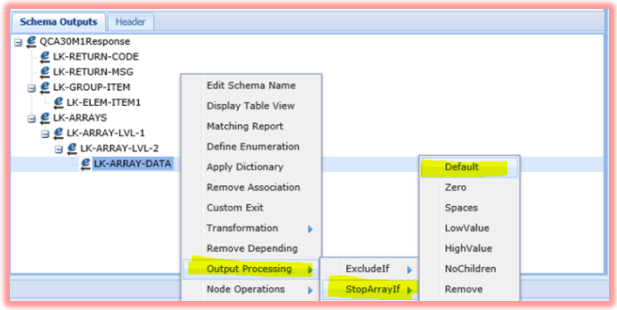


Figure 2:



In Figure 3 below all other StopArrayIf processing would be chosen as usual. In the case below a StopArrayIf condition of 'Default' is chosen to stop all processing when the array has encountered no more data.

Figure 3:



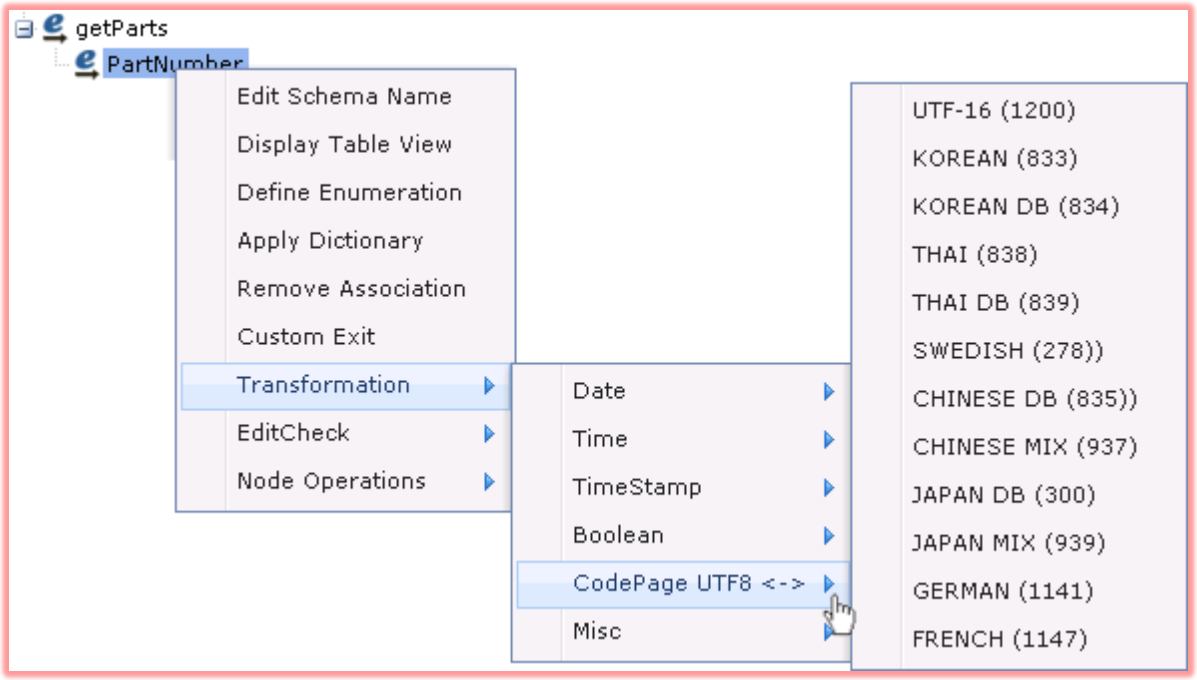
- Remove: removes the stop-array condition.

Remove Association: removes all associations from the target node.

Remove Depending: removes an occurs-depending-on association.

Transformation: allows you to change the format of the item's properties (e.g. change date from YY-MM-DD to MMDDYY or "true or false" to "Y or N"). There are several formatting choices for each available property type. This functionality is present in inbound and outbound web services.

- **Date:** change date format (e.g. YYMMDD to YY-MM-DD, etc.).
- **Time:** change time format.
- **Timestamp:** change timestamp format.
- **Boolean:** change Boolean value format (e.g. true/false to T/F, etc.)
- **CodePage UTF8:** choosing this menu option allows you to choose codepage conversion to and from UTF8 (variable length characters) to DBCS (two bytes per character). SOLA uses z/OS Conversion Services to do the conversions, so you'll need to have that active on your system, and the appropriate codepages will need to be installed. Please see the SOLA Installation Guide and SOLA Administration guide for details.



- **Misc:** allows you to set miscellaneous settings:
  - **Retain XML cp37:** indicates that the data in this element is an application XML payload that the SOLA runtime must not parse and that has to be exchanged between the SOAP client and the application in EBCDIC format. This is valid for both input and output processing.

- **Retain XML UTF8:** indicates that the data in this element is an application XML payload that the SOLA runtime must not parse and that has to be exchanged between the SOAP client and the application in UTF-8 format. This is valid for both input and output processing.

SOLA offers the ability to validate the application's input and output data at runtime as a natural consequence of capturing the information required for transformation. See the Validation section on page 104 for details.

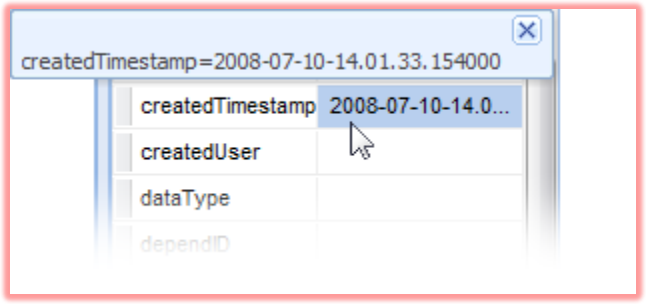
Unlink This Node: this will undo a drag-and-drop link operation (see page 115), unlinking a source tree item from an item in the target tree.

Analyzer Properties

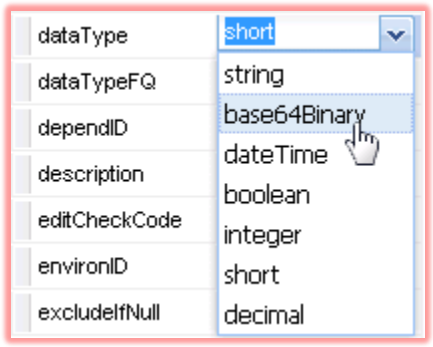
The properties displayed in the properties panel can be customized, however all SOLA installations are shipped with a standard set of properties that are described here. These properties not only display information about the selected item, they also contain configurable fields that can change the way the item, and consequently the web service, behave.

Not all properties appear for every field type.

If a property exceeds the available field space, double click on the field to display a pop-up dialog with the full length property value.



- **aType:** the datatype in an abbreviated form.
- **columnNm:** this is the name of the legacy tree item.
- **ctxSnstiveID:** this is ID of the legacy tree item that is linked to the displayed schema tree item
- **dataType:** indicates the item's data type. Legacy item data type names indicate mainframe data types while the Schema item data type names conform to open system data types. This is a drop down list and the value can be changed. See Appendix A: Schema and Copybook Generation on page 265 for details about the available options.



- **dependID:** if the selected item is part of an 'occurs depending on' array, this field displays the id of the item it depends on.
- **description:** a free form description of the item. You can use this to facilitate reuse by making it easier for others to understand your analysis.
- **effective:** timestamp that indicates when the item was created and made effective
- **environID:** ID corresponding to the Environment in which the analysis is being done
- **excludeifNull:** this item is a drop down menu with two values, Y and N. Select Y to filter responses based on their "natural nullable state" (e.g. 0 for numeric items or an empty string for strings). If Y is selected, the field will be excluded from the WSDL if it is null.

- **ID:** Unique internally assigned identifier for the element

- **io:** this is the variable's disposition (I/O status) and has the following options (the default value represents what SOLA believes to be most appropriate for the associated variable):

    - **I:** indicates that the variable is contained in the SOAP request and is input to the COBOL or PL/I program.
    - **O:** indicates that the variable is output from the COBOL or PL/I program and will be published in the SOAP response.
    - **X:** indicates that the variable is excluded, which means it is not referenced or usage of the variable is unknown by the COBOL or PL/I program and will not be part of either a SOAP request or a SOAP response.
    - **B:** indicates the variable is contained in the SOAP request and is input to the COBOL or PL/I program and is output from the COBOL or PL/I program and will be published in the SOAP response.

- **len:** the maximum physical length, in bytes, of the variable. This will typically be the same as the Precision, though in the case of decimal data types, Length and Precision will be different.

- **level:** this is the variable's level within the COMMAREA. This does not necessarily directly correspond to the level number in the commarea, with the exception of 88 levels. If the commarea structure level numbers are 01,05,10, the levels will be 1,2,3.

- **maxOccurs:** this is an Sitem property that indicates the maximum number of occurrences of an item. This item corresponds to an xml schema's maxOccurs value.

- **methodID:** this is the internal ID of the method

- **minOccurs:** this is an Sitem property that indicates the minimum number of occurrences of an item. This item corresponds to an xml schema minOccurs value.

- **namespace:** this is an optional Sitem property to set the namespace for the schema item

- **nodeType:** this is a menu with two options, e (Element) or a (Attribute). This field will determine whether the associated item is treated as an element or an attribute in the WSDL. When SOLA analyzes a compile listing, it determines what is input and output and attempts to set most output items as attributes for performance and efficiency reasons. Output Arrays are exceptions that will be represented as elements.

  The figures below show two results of a Quick Test on the same method. The first is with all fields set to A (Attribute) and the second is with all fields set to E (Element).

  All Fields set to Attribute:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--  StartTime = 2007-08-21-11.47.52.451 - EndTime = 2007-08-21-11.47.52.532 -
ElapsedTime = 81 milliseconds  -->
- <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 - <soap:Body>
  - <nameSearchResponse
      xmlns="http://nameSearch.ClientFinder.x4mlsoa.com/CA/SOLACA04/TXMLD990">
    - <Commarea ReturnCode="0" ReturnMessage="" ProgramVersion="1.0" SequelCode="0"
        CicsReturnCode="" HostSysid="CICB" TotalCenter="2" FetchCenter="2">
       <ClientInfo ClientName="HOGAN, RUTH S" ProducerId="7469968"
         ClientNumber="113340063" PhoneNumber="" />
       <ClientInfo ClientName="HOGAN, RUTH S" ProducerId="7469968"
         ClientNumber="987130063" PhoneNumber="" />
      </Commarea>
    </nameSearchResponse>
  </soap:Body>
</soap:Envelope>
```

All Fields set to Element:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--  StartTime = 2007-08-21-11.49.41.858 - EndTime = 2007-08-21-11.49.41.924 -
ElapsedTime = 66 milliseconds  -->
- <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 - <soap:Body>
  - <nameSearchResponse
      xmlns="http://nameSearch.ClientFinder.x4mlsoa.com/CA/SOLACA04/TXMLD990">
    - <Commarea>
       <ReturnCode>0</ReturnCode>
       <ReturnMessage />
       <ProgramVersion>1.0</ProgramVersion>
       <SequelCode>0</SequelCode>
       <CicsReturnCode />
       <HostSysid>CICB</HostSysid>
       <TotalCenter>2</TotalCenter>
       <FetchCenter>2</FetchCenter>
      - <ClientInfo>
         <ClientName>HOGAN, RUTH S</ClientName>
         <ProducerId>7469968</ProducerId>
         <ClientNumber>113340063</ClientNumber>
         <PhoneNumber />
        </ClientInfo>
      - <ClientInfo>
         <ClientName>HOGAN, RUTH S</ClientName>
         <ProducerId>7469968</ProducerId>
         <ClientNumber>987130063</ClientNumber>
         <PhoneNumber />
        </ClientInfo>
      </Commarea>
    </nameSearchResponse>
  </soap:Body>
</soap:Envelope>
```

- **NSalias:** this is an optional Sitem (Schema tree item) property to set the namespace alias to correspond to the namespace setting on the schema item.

- **objectType:** this is the object's type, either Citem (Legacy tree item) or Sitem.

- **occurs:** for arrays, this will be the number of rows contained in the array and will only be populated at the group level. For the variables within the array, this field will be zero.

- **occursDepth:** this relates directly to the array's dimensions. In a multi-dimension array, for instance, variables in the first dimension will be at 'Occurs Depth' one. Those in the array's second dimension will be 'Occurs Depth' two, etc.

- **occursFrom:** this is the starting point of the array in the legacy program (e.g. a 1-100 array will have an occursFrom of 1). The minOccurs value may initially gets its value

from occursFrom, if present. If there is no occursFrom, the value of minOccurs will be defaulted.

- **occursSize:** this is the total length of all data items that comprise a single row of the COBOL array. If this number was multiplied by the 'Occurs' value it would yield the total length of the COBOL array.
- **offset:** this is the offset (relative to zero) of the data item within the overall data structure.
- **parm:** Not used Commarea Analysis
- **pattern:** Not used Commarea Analysis
- **precision:** the data variable's precision. This will typically be the same as the len (length), though in the case of decimal data types, len and precision will be different.
- **programID:** this is the internal ID of the program
- **redefID:** if the item redefines another item, this field displays the internal ID of the redefining item.
- **resultSet:** Not used Commarea Analysis
- **rowNum:** for Citems (legacy tree items), this represents their row number in the legacy tree. For Sitems (schema tree items), this represents their row number in the schema tree.
- **scale:** for some datatypes (such as fractions), the scale represents the number of significant positions after the decimal point
- **schemaNm:** this represents an Sitem's (schema tree item) name in the WSDL.
- **specialCond:** Not used Commarea Analysis
- **stopArrayifNull:** this is a dropdown menu that gives you the ability to pick an elementary item within an array to use as a sentinel to stop table processing if the item's value is null ("natural nullable state" for that particular data type, e.g. 0 for numeric items or an empty string for strings). The Stop Array column will contain a checkbox if the field is part of an array (otherwise it will be empty). Select S to activate stop array processing for the associated field(stop the array if null). Select N to disable stop array processing (not stop array if field is null).
- **Type:** displays the Citem's picture clause, which will either be a fixed value or a drop down menu containing two or more options. The available options depend on the data type. The following is a compilation of all possible options:
  - **Grp:** indicates that the variable is a group level variable within the data structure.

- **Dis:** indicates a display, or character (PIC X) variable.
- **Num:** indicates a numeric (PIC 9) variable.
- **Bin:** indicates a binary (comp) variable.
- **Pck:** indicates a packed decimal (comp-3) variable.
- **Ned:** indicates a numeric edited variable.
- **Ptr:** indicates a pointer variable.
- **B64:** indicates to SOLA that this variable's data must be converted to Base 64 format before being sent in a SOAP response or converted from Base 64 to native binary if received as part of a SOAP request. This is used for transporting binary data such as photographs or PDF files for instance.

- **value:** this field is present in both Citems and Sitems, though it can only be set for Sitems. Setting the value of an Sitem hard codes that value in the WSDL. This means that the web service will always use the value you set and will never take input for this item from the consumer.

Using SOLA Developer – Callable APIs and Containers

SOLA offers three methods of passing a structured block of data to a program; Commarea, CICS Channels and Containers and Callable APIs. The traditional DFHCOMMAREA is limited to 32k, so to overcome this limit, SOLA can create web services by exploiting CICS Channels and Containers and Callable APIs.

Callable API and Container programs are imported and analyzed using the Commarea analyzer. Please read the Commarea (inbound, bottom up) section to familiarize yourself with the import and analysis processes before continuing. This section will highlight the differences between creating a web service from a standard Commarea program and callable API and container programs.

Callable APIs

A Callable program is invoked using a COBOL CALL instruction. Callable programs use the standard register linking convention. The advantage to using callable programs is that there is no limit to the size of the data area that is passed.

There is a pre-requisite condition to enable callable programs. In order for a web service created from a callable program to function, a SOLA runtime Callable plugin module must be running locally in the region in which the application callable program is executed

There are two SOLA runtime Callable plugin module that are delivered and the choice of which module to use is as described below:

XMLPC205 - If your application callable program is coded **not** to expect DFHEIBLK as first parameter then use this module.

XMLPC206 - If your application callable program is coded to expect DFHEIBLK as first parameter then use this module.

Step 1 – Mainframe Preparation

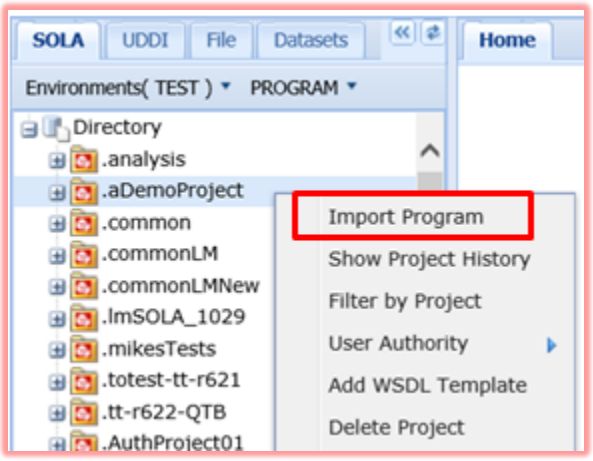
Preparation steps are identical to that of importing Inbound Commarea programs

To work with Callable API programs a PPT entry is needed in the WOR region that points to the AOR region. Since the Callable API programs are not called directly but instead run under the control of SOLA, the PPT entry for a DOM API program must specify REMOTENAME(XMLPC205) or REMOTENAME(XMLPC206).

WOR	**AOR**
PPT: DEFINE PROGRAM(yourCallprogName) LANG (LE) STATUS (Enabled)	**PPT:** DEFINE PROGRAM(XMLPC205) LANG (LE) STATUS (Enabled)

REMOTE REGION(yourRegion) REMOTE NAME:XMLPC205 REMOTE TRANID: (yourTranId)	
	PCT: DEFINE TRANSACTION(yourTranId) PROGRAM (DFHMIRS) PROFILE (DFHCICSA) STATUS (Enabled)

Step 2 – Importing a Callable Program

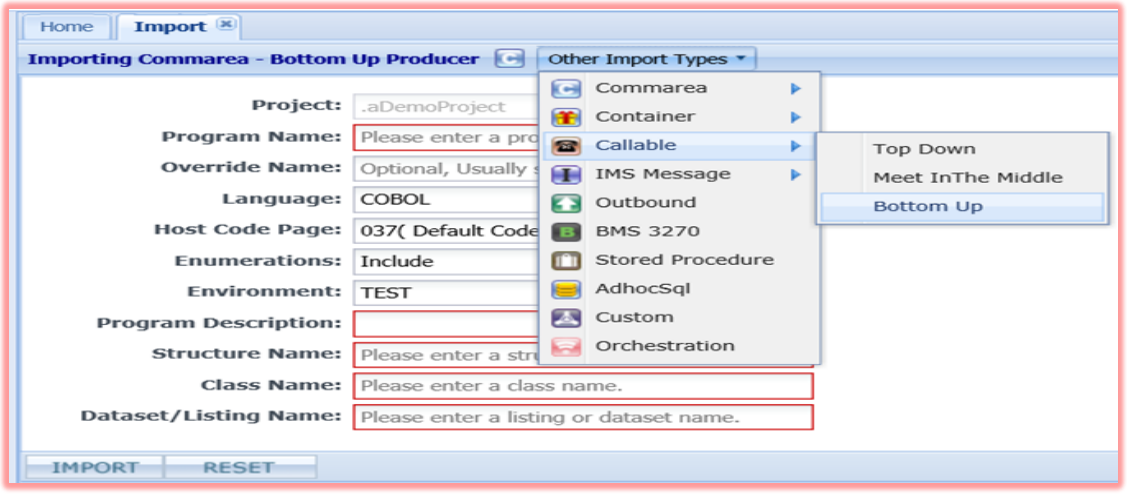


Callable API import and analysis is identical to that of Commarea programs (see page 45). It is important, however, that you use the correct import panel. Even though the panels may appear identical, there are backend differences.

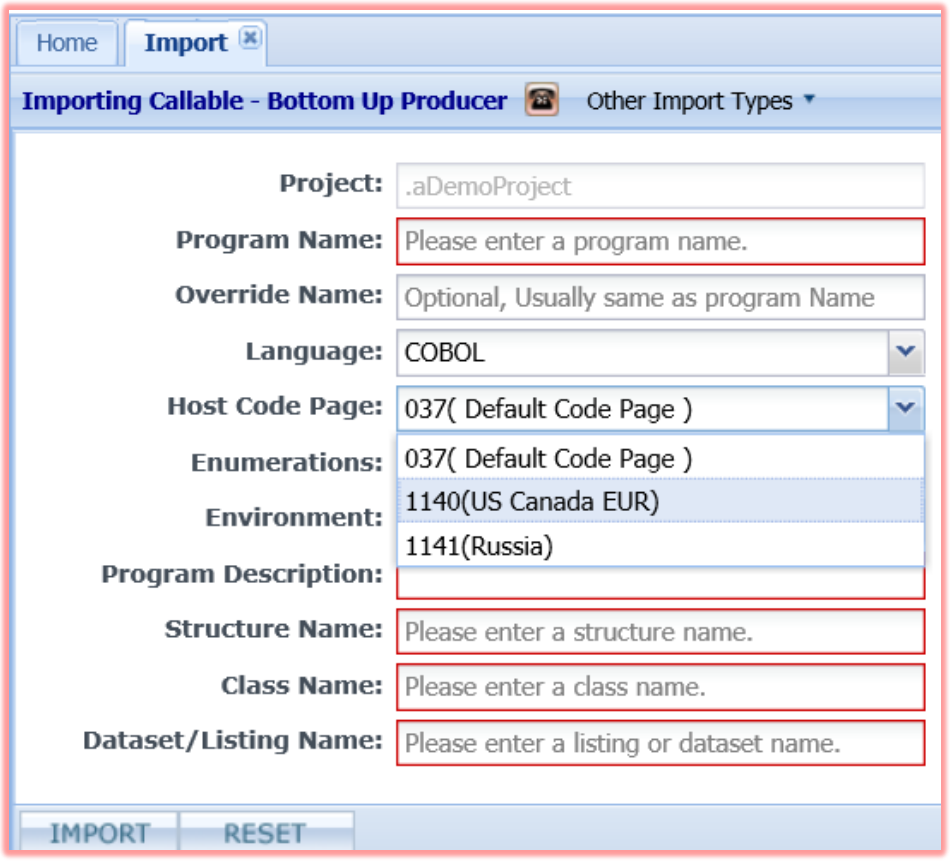
To get to the callable import panel, select the project you wish to import to and right-click it. From the pop-up menu, select **Import Program**. If you wish to import the program to a new project, first follow the steps for creating a new project on page 22.

After you select **Import Program**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

The default program type is commarea, so use the **Other Import Types** menu to select "Callable", then select one of the three import types (bottom-up, meet-in-the-middle or top-down), just like commarea. In this example we are selecting **Bottom Up**.



The Import panel will change to display the callable import panel.



From this point, follow the steps for importing a Commarea program (matching the type you selected, bottom-up, etc.) to create a web service from a callable program. Note the **Host Code Page** selection options (*this is described further in the SOLA Administration Users Guide and in the 'Importing a Commarea Program' and 'Admin Menu' button sections of this guide*.)

Channels and Containers

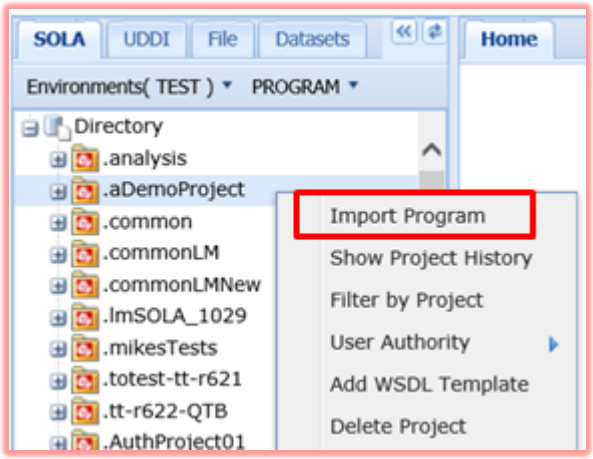
Channels and containers are programs that implement the channel and container interface to overcome the 32K Commarea limit.

Channels and containers are a feature of CICS TS 3.1 (and above).

Step 1 – Mainframe Preparation

Preparation steps are identical to that of importing Inbound Commarea programs.

Step 2 – Importing a Channel/Container

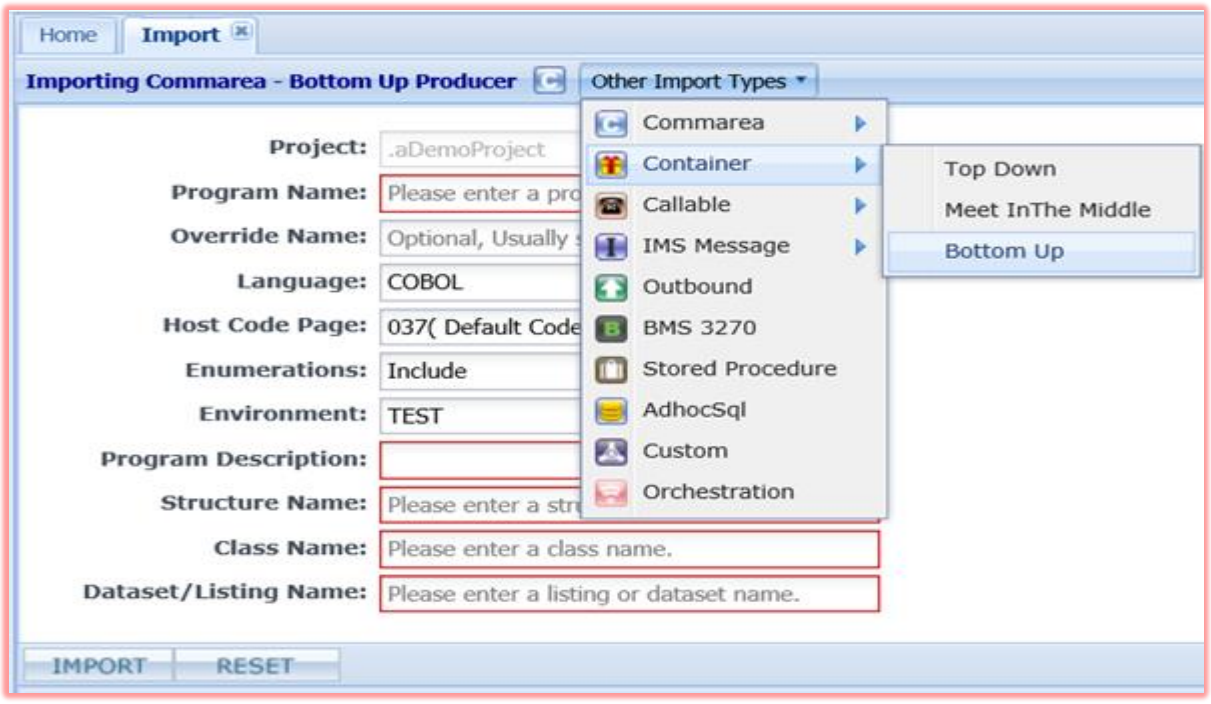


Channel/Container import and analysis is similar to that of inbound bottom-up Commarea programs (see page 45). The only difference is the addition of several fields to the import panel that describe the channel and containers used by the program.

To get to the channel/container import panel, select the project you wish to import to and right-click it. From the pop-up menu, select **Import Program**. If you wish to import the program to a new project, first follow the steps for creating a new project on page 222.

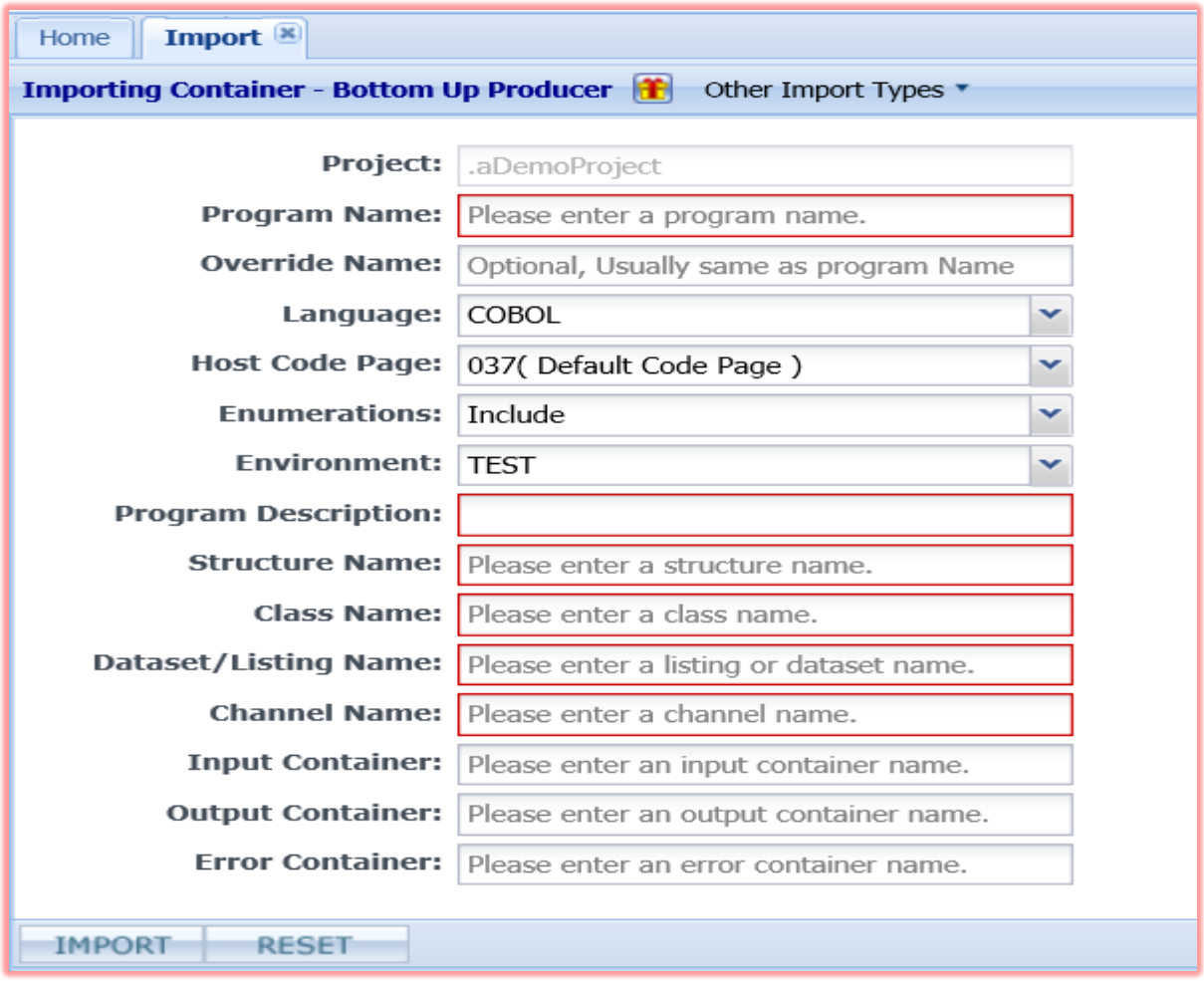
After you select **Import Program**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

The default program type is commarea, so use the **Other Import Types** menu to select "Container".



If you select Top Down or Meet in the Middle, you will be taken to the WSDL import screen. From that point, you can follow the instructions for importing the same type of commarea program.

If you select Bottom Up, the Import panel will change to display the channel/container import panel, which is a little different from the Commarea import panel. See the illustration below:



This panel contains some fields not present on the standard Commarea import panel.

- **Channel:** the name of the mechanism with which the program's containers are associated with. A channel is analogous to a COMMAREA, but it does not have the constraints of a COMMAREA.
- **Input Container:** the storage structure that contains input to the program.
- **Output container:** the storage structure into which the programs sends its output
- **Error container:** the storage structure into which the program sends its error responses.

Fill out the extra fields described above, then follow the steps for importing an inbound Commarea program using bottom-up methodology to create a web service from a channel/container. Note the **Host Code Page** selection options (*this is described further in the SOLA Administration Users Guide and in the 'Importing a Commarea Program' and 'Admin Menu' button sections of this guide*.)

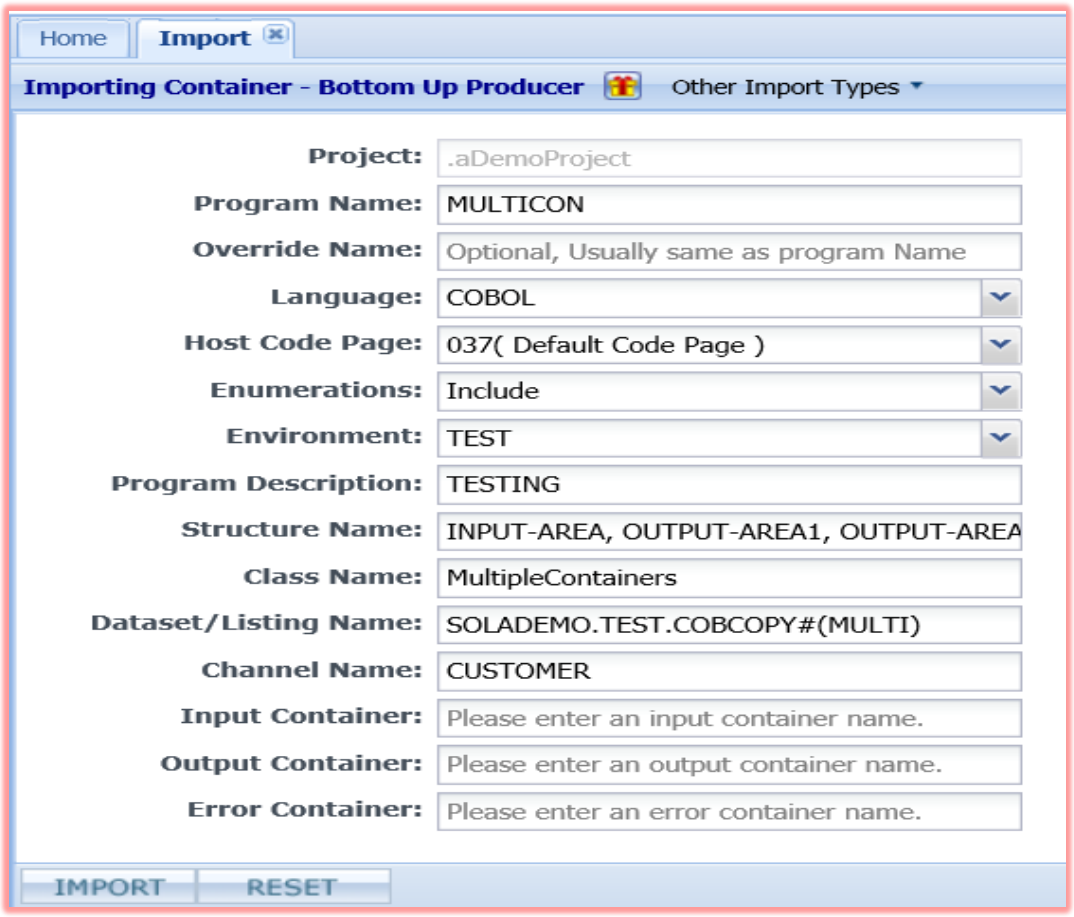
Note: If your program doesn't follow the simple input container, output container and error container approach, just enter the channel name and leave the container names blank. You can enter them later.

Specifying your Container Names

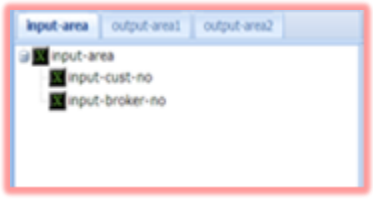
If your program supports a more complex arrangement than the approach that SOLA provides on the initial Analysis screen (default input container, output container and error container), you can specify them during analysis.

Let's say you have a program called MULTICON that uses three containers, INPUT-AREA, OUTPUT-AREA1 and OUTPUT-AREA2. Each container is associated with a 01 structure, and you would:

- Enter the three 01 level structure names in the **Structure Name** field (separated by commas)
- Enter the Channel name
- Leave the Container names blank



With three containers, the Analysis screen would show three tabs in the source area, one for each of the 01 level Structure names.

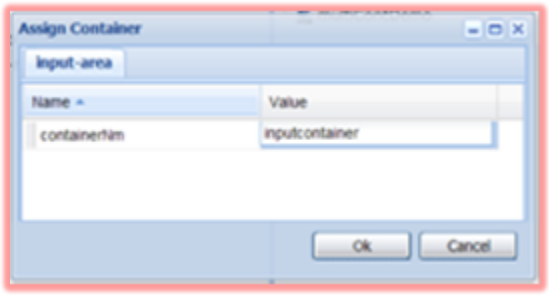


You can associate a container with a structure by choosing the tab for that structure, and then right clicking on the structure's root (the 01 level name in the tree), and choosing Assign Container from the pop-up menu.

Right click on the 01 level structure name (the root of the tree) to bring up a pop-up menu of choices for that structure, and choose **Assign Container** from that menu.

This will bring up a pop-up panel for you to enter the container name for the structure.

Note: The same menu pops up no matter which of the nodes you click on in the structure tree. However, **Assign Container** will only correctly associate a container with a structure when you click on the 01 level (root) of the structure.



There's no need to specify whether a container is Input or Output, SOLA will determine that based on whether you drag structure elements to the input schema or output schema (or both).

Note: The Container name you are assigning should match the Container name used in the originating program that created the WSDL that was input into this meet in the middle Import.

Creating an Inbound Channel and Container Web Service from a WSDL and a Program – Meet-in-the-Middle

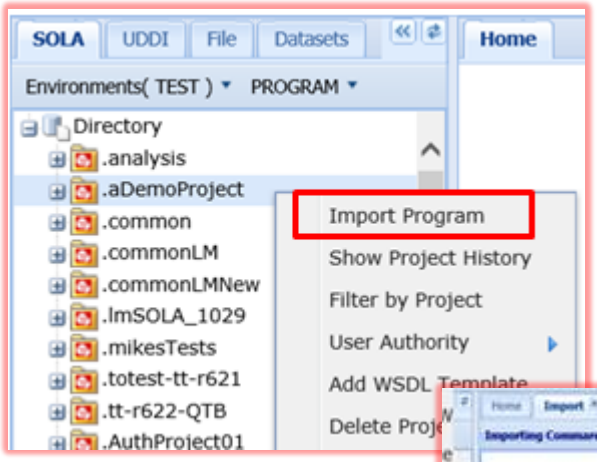
It is often the case that your corporate architects want you to conform to a standard service architecture, and they'll expect you to do so with a WSDL that they provide. When you have a program that fairly closely matches their requirements you can merge the WSDL and the program using SOLA's Meet-in-the-Middle approach, allowing your existing program to interact using the WSDL service definition.

Step 1 – Mainframe Preparation

The only mainframe preparation required with the meet-in-the-middle approach is for you to identify a copybook that describes the program's interface (typically DFHCOMMAREA, but for example, it could be any name of a typical COBOL copybook beginning with a 01 level data name). You will need to map each of the WSDL fields to the copybook fields by dragging and dropping each one from the left to the right. The final step will be to click the Finalize button. It is at this point that SOLA will create a template containing those mapping rules. This is explained in further detail in the pages that follow.

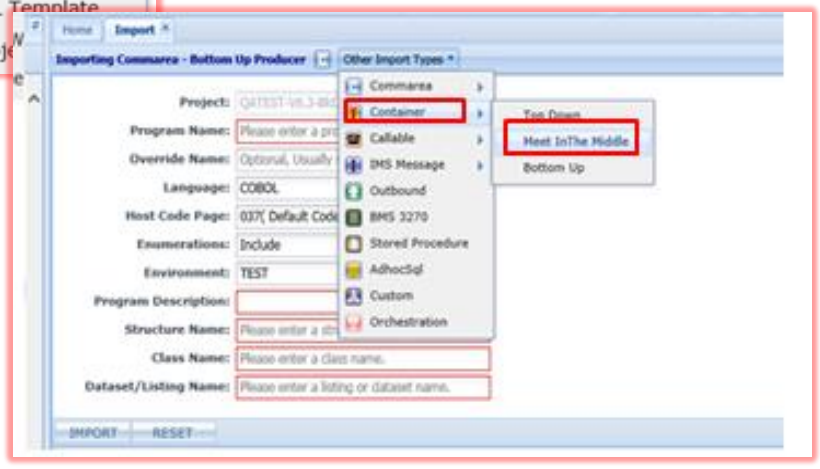
Step 2 – Importing a Channel/Container

To get to the channel/container import panel, select the project you wish to import to and right-click it. From the pop-up menu, select **Import Program**. If you wish to import the program to a new project, first follow the steps for creating a new project on page 21.



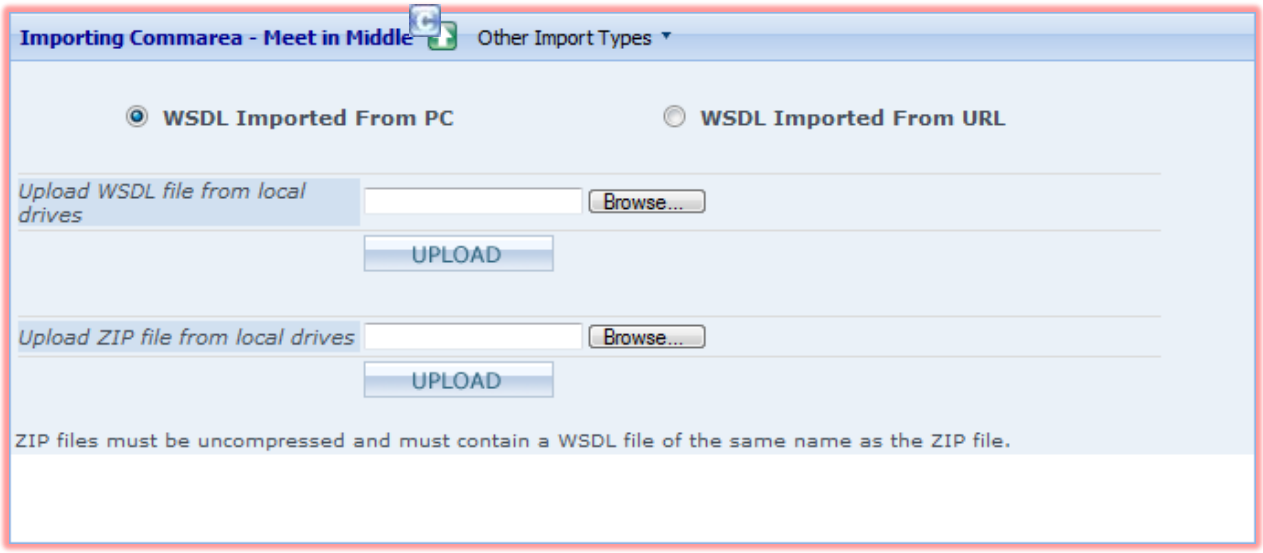
After you select **Import Program**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

The default program type is Commarea. You will select the **Other Import Types** menu to select "**Container**".



Next select **Meet in the Middle** and you will be taken to the WSDL import screen

The Import panel will change to display the WSDL import panel.



This panel provides the means to import a WSDL file into SOLA from a location on the Internet/Intranet or on your local machine. To import a WSDL file from your local machine, either type the full path to the WSDL in the **Upload WSDL file from local drives** field or click the **Browse…** button to locate the WSDL using Windows Explorer.

To import a ZIP file, follow the same process described above, but use the **Upload ZIP file from local drives** field instead. SOLA does not support compressed ZIP files, so make sure the ZIP file you are uploading is uncompressed. This option is for the importing of WSDL files that utilize external references. The ZIP file must contain a WSDL file of the same name as the ZIP file and all files referenced by the WSDL.

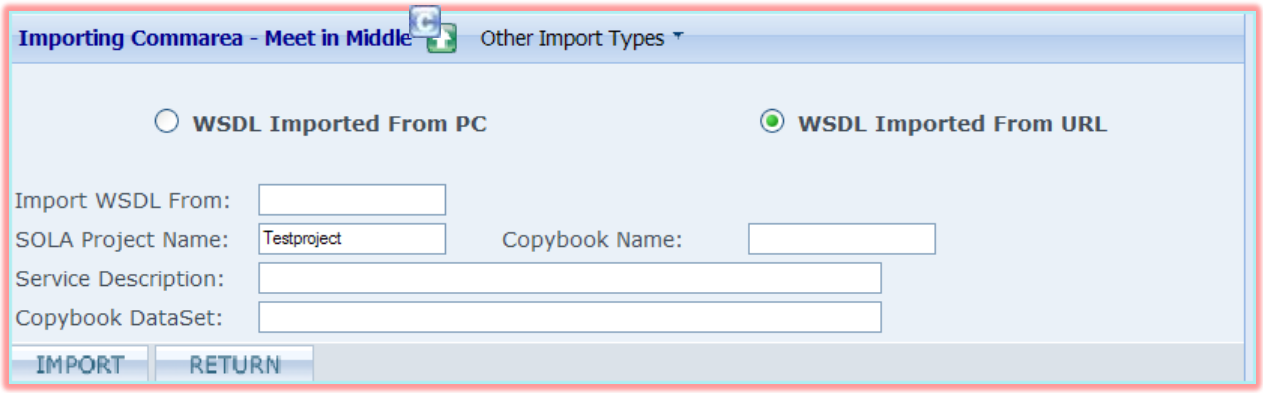
For example, if the ZIP file is called abc123.zip, then it must contain a WSDL file called abc123.wsdl and all files referenced by abc123.wsdl.

Once you have made your selection, click 

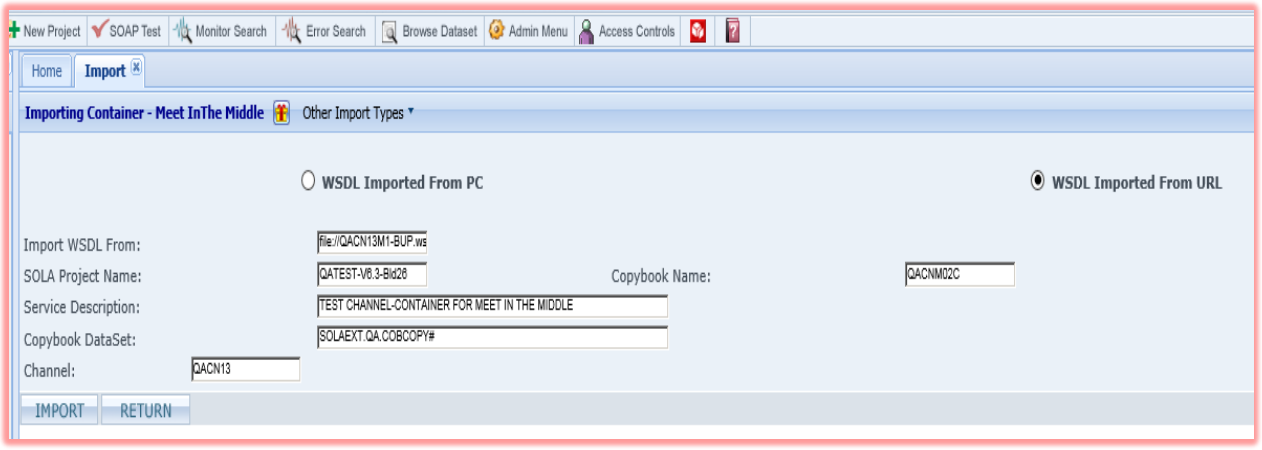
. Make sure you are clicking the correct Upload button (there are two).

If you want to upload the WSDL from a URL (internet/intranet) click 

. Either way, you will be taken to the following panel.



Fill in the fields as required:



- **Import WSDL From:** the address (URL) of the WSDL file being imported. If you chose to import from the local machine, the path of the local file will be displayed there. If you chose to upload from a URL, copy and paste (or manually enter) the URL into this field.

  Note: The provided WSDL must contain a SOLA compliant soap action as seen in the example below. Only include the codepage if other than the default codepage 37 is used.

```
- <binding name="CLASS_QACA991PBindingName" type="tns:CLASS_QACA991PPortTypeName">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  - <operation name="QACA991M">
      <soap:operation style="document" soapAction="/CA/QACA99P/QACA991T/CCP:UTF-8/HCP:1140"/>
    - <input>
        <soap:body use="literal"/>
      </input>
```

- **SOLA Project Name:** the name of the project under which the WSDL file will be imported. This is pre-populated and cannot be changed.

- **CopyBook Name:** specify the member name where SOLA is to find the interface definition (copybook) for your program.

- **Service Description:** a brief description of the service that will be created from the WSDL file.

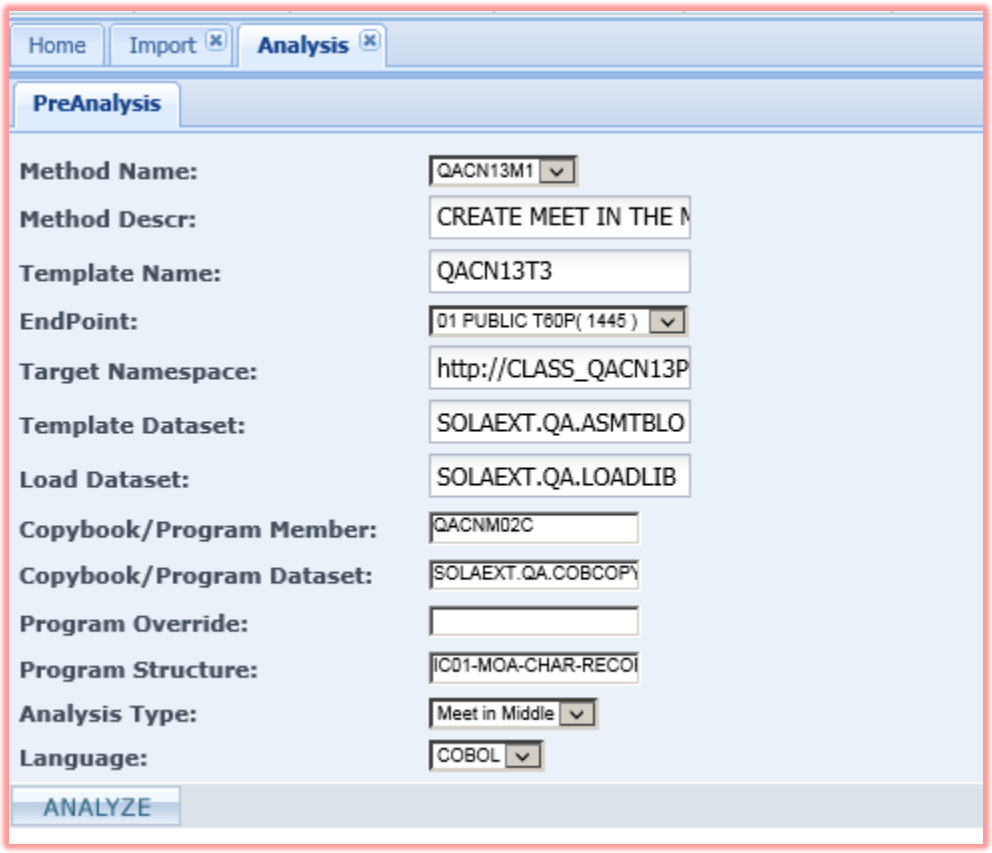
- **Copybook DataSet:** specify the Copybook DataSet (a PDS) where SOLA will find the interface definition (copybook) that will be used for your program. WSDL attributes and elements will be mapped to these copybook fields by you.

- **Channel:** the name of the mechanism with which the program's containers are associated with. A channel is analogous to a COMMAREA, but it does not have the constraints of a COMMAREA.

When you are ready to continue, click. IMPORT

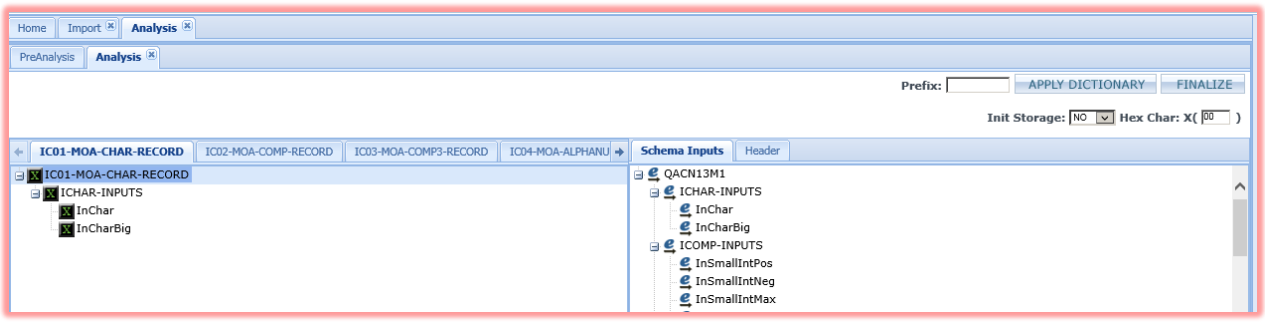
To return to the previous panel if you need to start over or locate data to enter into one of the panel fields, simply click RETURN.

After clicking IMPORT the Analysis tab and PreAnalysis panel will appear. The input requirements are exactly the same as those used for Commarea – Meet In The Middle.

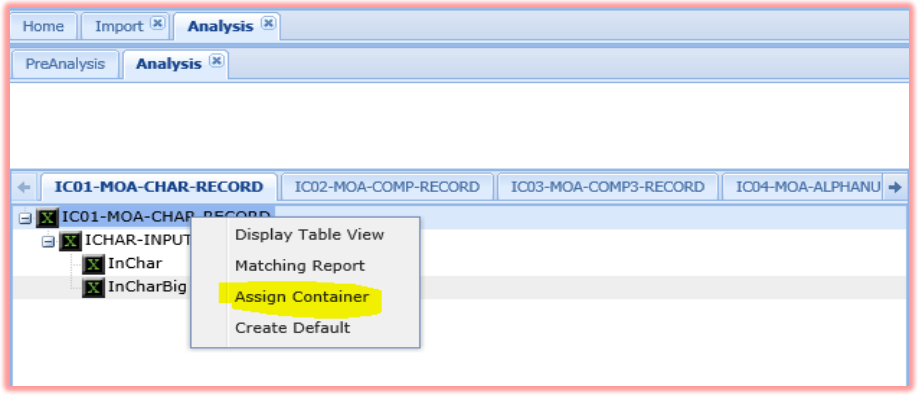


When you have filled out the required fields, click the **ANALYZE** button.

The meet-in-the-middle analyzer is identical to the bottom-up analyzer, the Schema tree is on the right, and the Legacy tree is on the left.



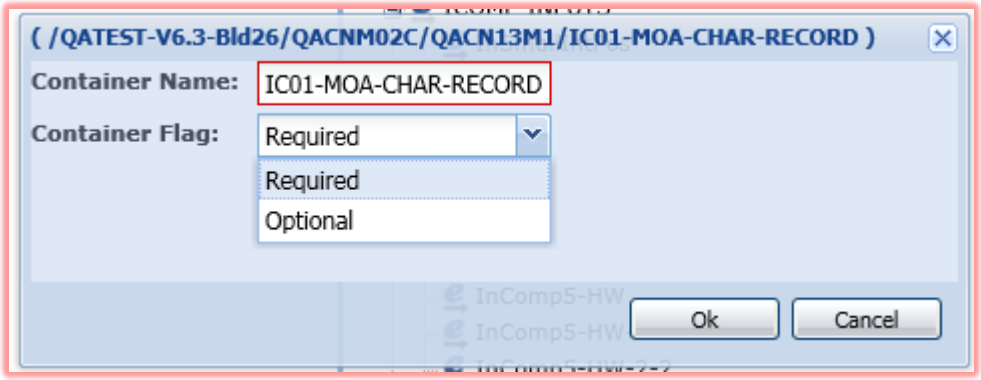
You can associate a container with a structure by choosing the tab for that structure, and then right clicking on the structure's root (the 01 level name in the tree), and choosing **Assign Container** from the pop-up menu.



Enter the Container Name you are assigning the structure to and click **Ok**.

Note: You must use the same Container Name from the originating program that created the WSDL you used in the Import step above.

In our example all Containers are Required, but you may also want to have a structure be assigned as an Optional Container.

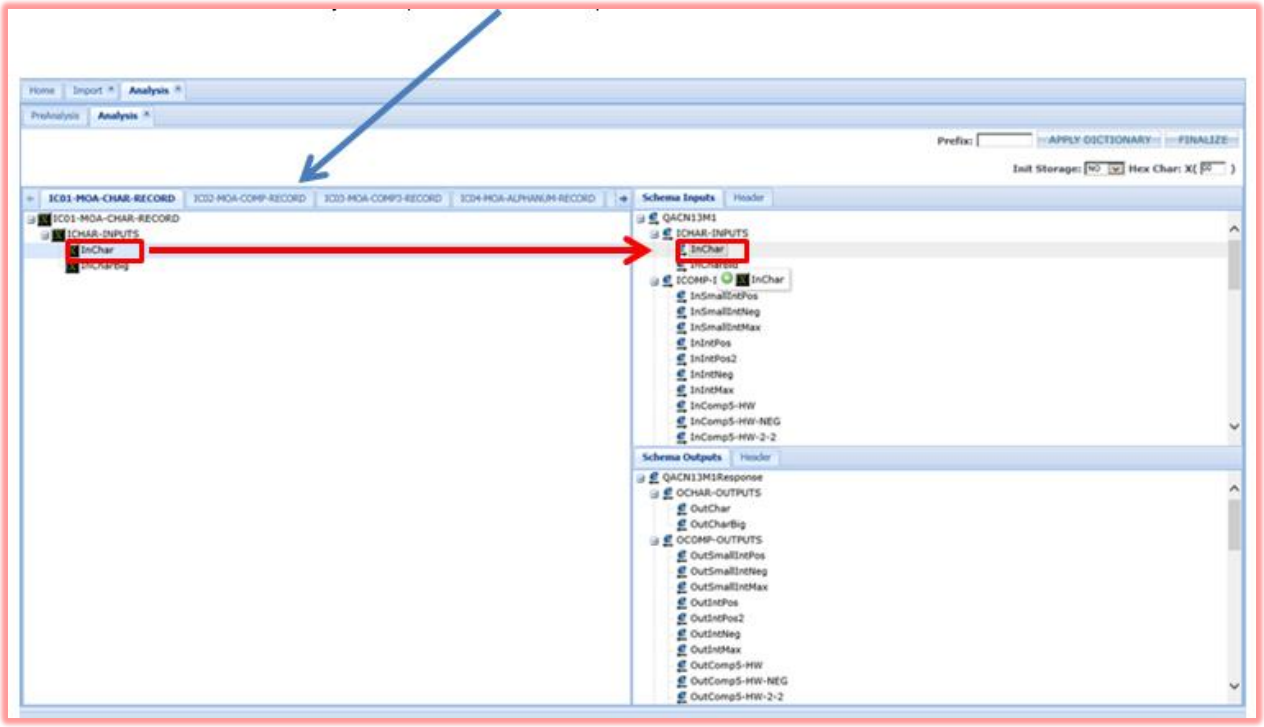


Your task now is to match the elements in each tree, so that SOLA can perform the necessary transformations between the two structures. To do so, you must link every component in each tree to a matching item in the other tree by dragging and dropping each data element to the schema input and output.

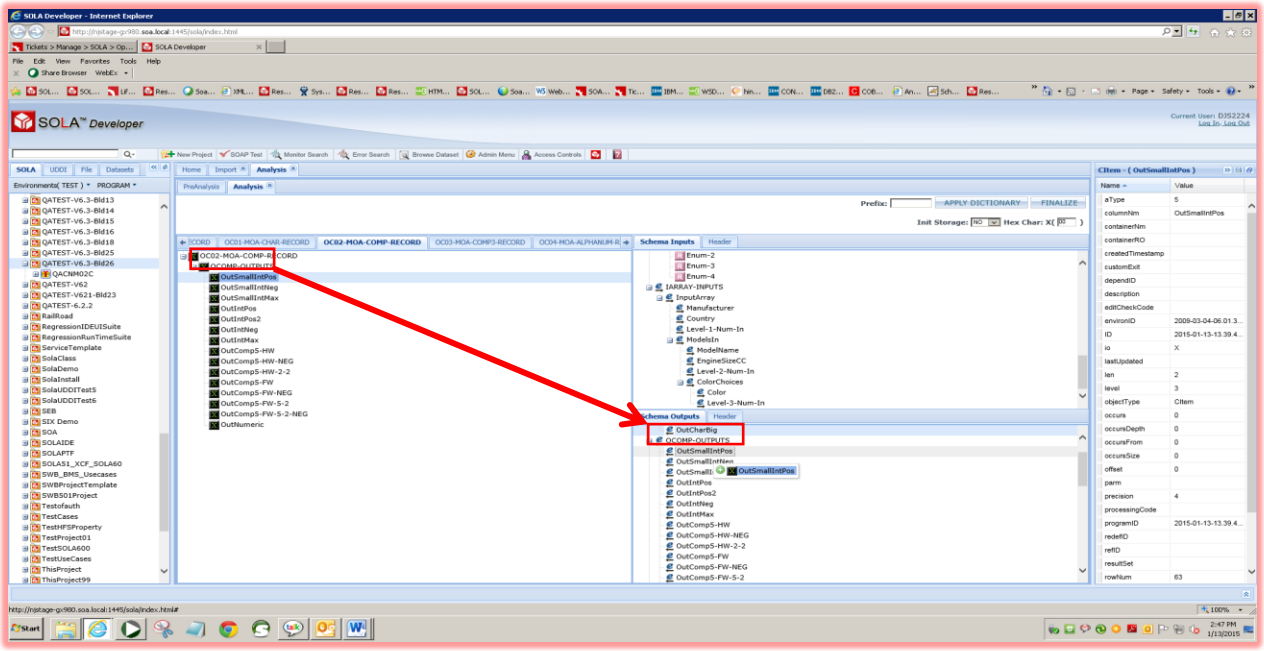
In the following illustration we have begun by dragging each input or request element on the left legacy side of the workspace to its matching schema item on the right. You must also include the Group level item; in this case ICHAR-INPUTS.

In this example there are five input containers shown on the left side of the workspace followed by five output containers.

Each container will have its own Tab in the workspace (IC01 thru IC05 and OC01 thru OC05).



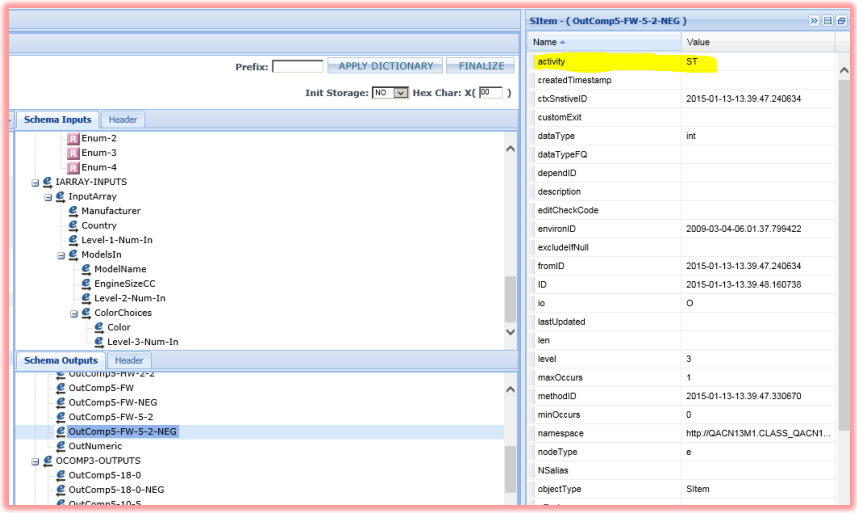
Continue mapping (dragging and dropping) the output containers to the Schema Outputs:



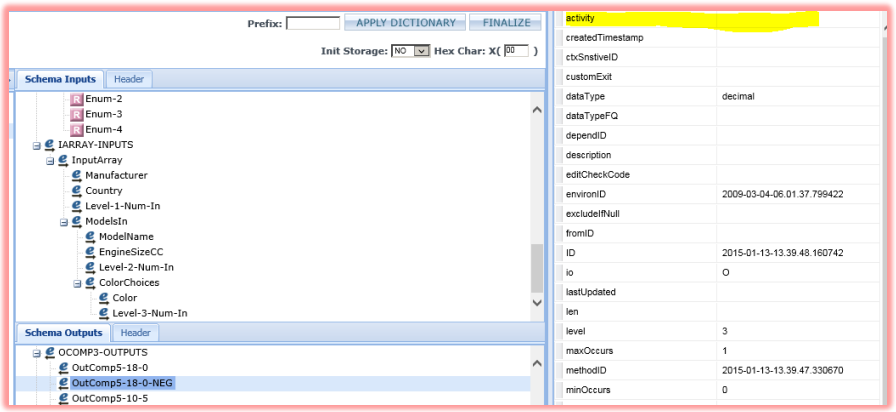
Note: You will see an icon when you have landed on the schema field you wish to drop the legacy element onto.

Note: To provide a visual 'schema-to-structure' mapping you can **click** on any schema field on the analysis screen and the corresponding legacy structure field will **flash** providing a visual cue to the developer about the field mapping.

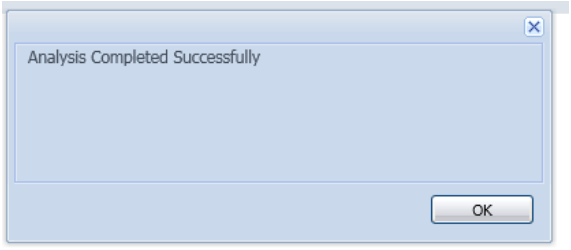
When a schema field has been successfully mapped you will see the SItem property value set in the property panel.



If the schema field was not mapped properly the property value would be blank:



Click FINALIZE to complete the analysis. You will be presented with a confirmation dialog indicating the analysis was successful.



Using SOLA Developer – IMS

Creating a Web Service from an IMS Program – Bottom Up

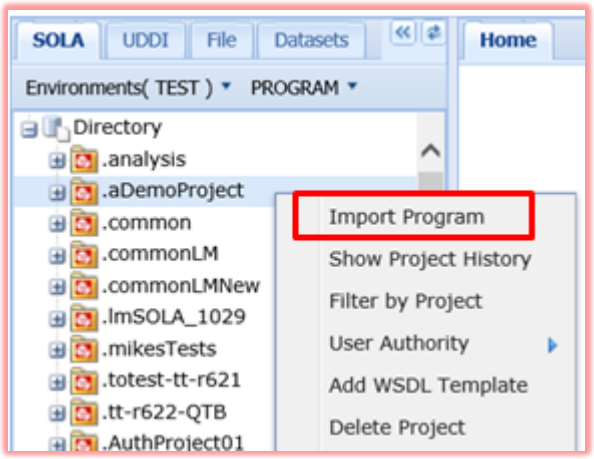
Creating a bottom-up web service from an IMS program is very similar to creating a web service from a Commarea program, but there are some fundamental differences that must be understood.

If you have not already read the section on creating web services from Commarea programs, it is suggested that you do so, as you will need all of the knowledge contained in that section to understand how to use the IMS analyzer.

Step 1 – Mainframe Preparation

For an IMS program you'll need the PDS member or members that contain the IMS input and output segment copybook(s) (a compile listing of your program works equally as well). You'll also need a PDS to store the generated template, and a loadlib for the link-edited version of the template. Finally, your target SOLA container must be configured to connect to IMS using OTMA or IMS Connect.

Step 2 – Importing the Program

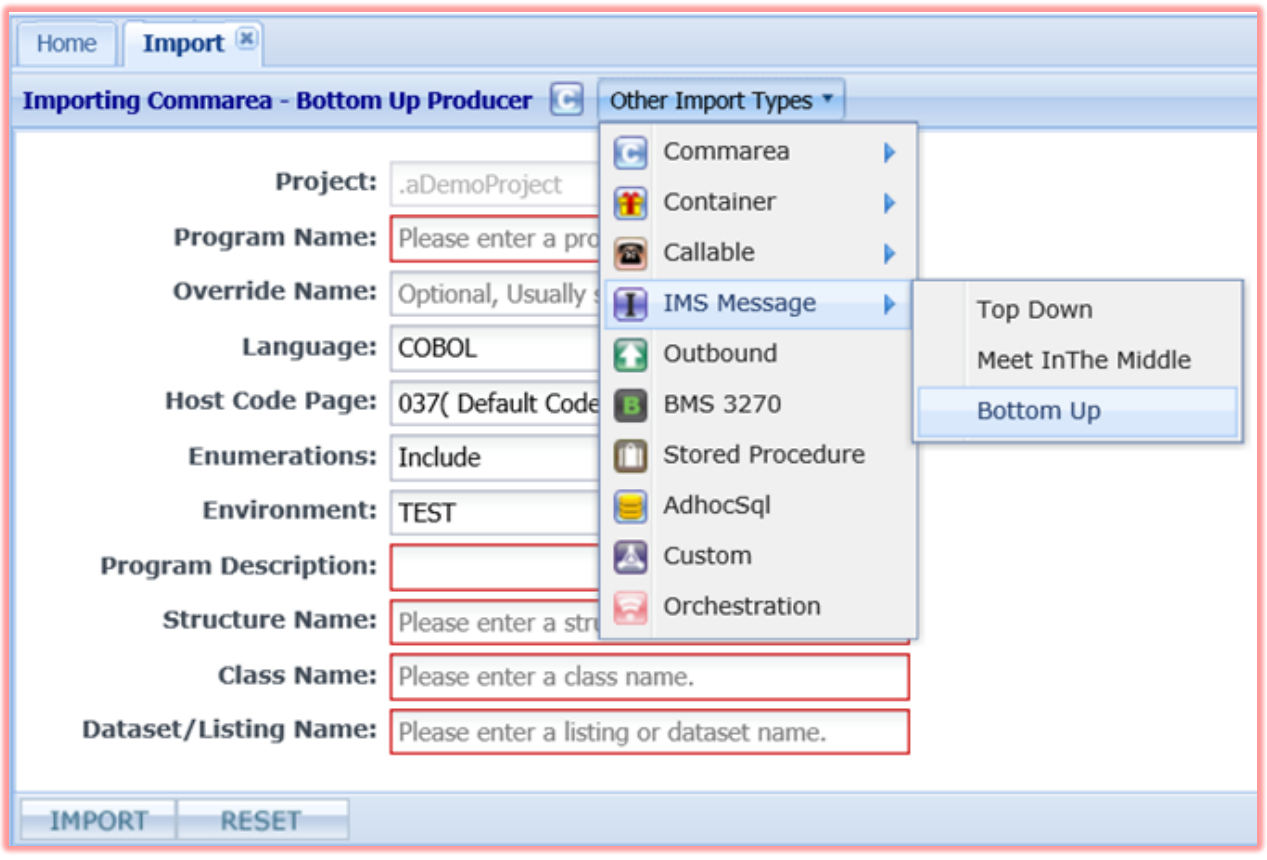


Select the project you wish to import to and right-click it. From the pop-up menu, select **Import Program**. If you wish to import the program to a new project, first follow the steps for creating a new project on page 22.

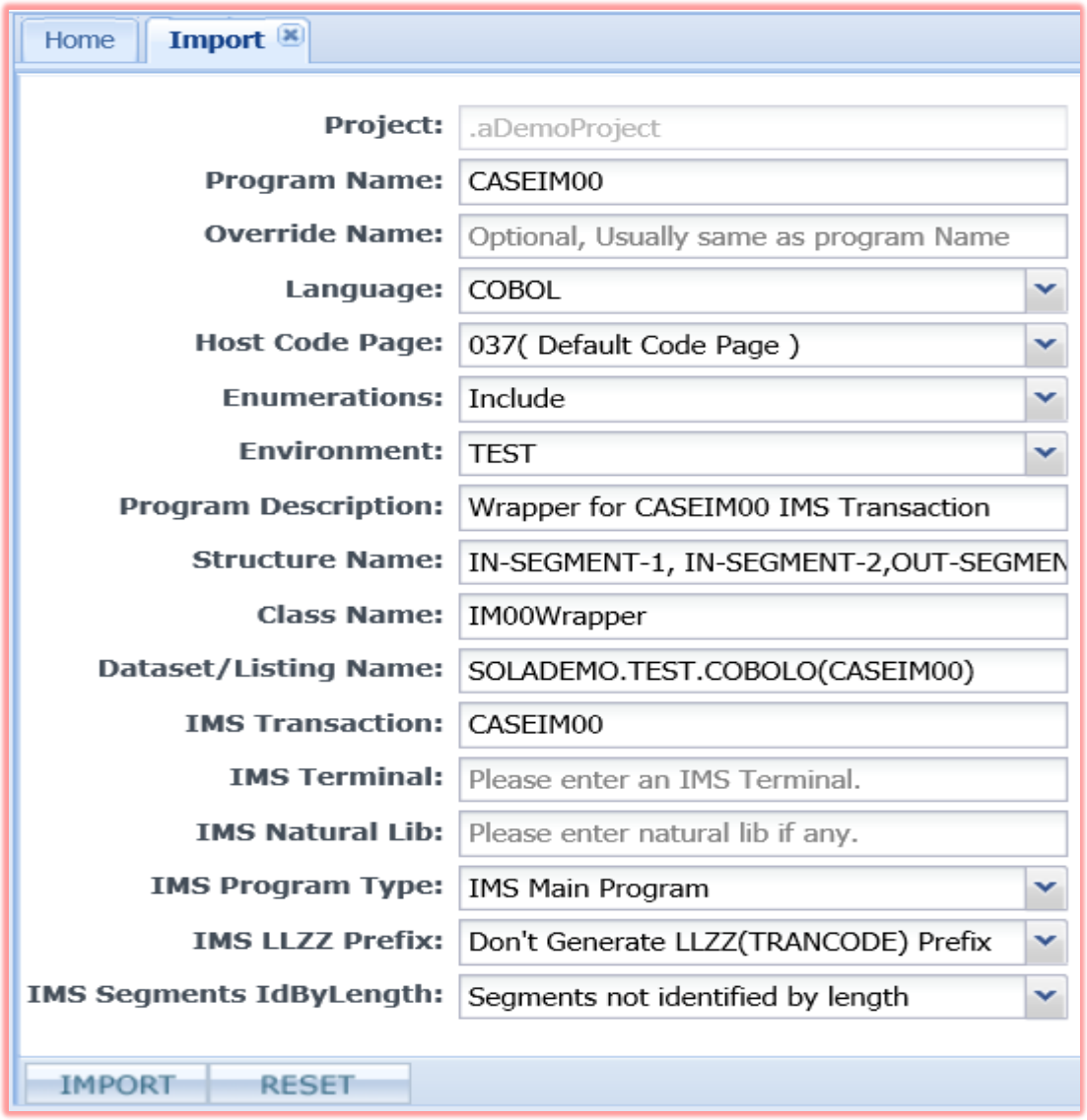
In order to import a program into a project, you must be an authorized user of that project. Once the program is imported, you can drag and drop the program from one project to another. However, you must also be an authorized user of the project you wish to move the program into.

After you select **Import Program**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

Using the **Other Import Types** menu, select **IMS Message**, then from the sub-menu, select **Bottom Up**.



The Import panel will change to display the IMS bottom up import panel as outlined below:



The IMS Message Import panel consists of a series of fields used to provide information about the source program and the destination SOLA program that will be created.

Fields outlined in red are required. The red outline disappears when the field is populated.

- **Project:** this field is pre-populated and contains the name of the project into which the program is being imported. Although it cannot be changed during import, you can drag the program into a different project after it has been imported.

- **Program Name:** For IMS plugin that is capturing IMS Main progarm, this is just a name on the SOLA directory to which this service is being captured into. If you are capturing IMS Subroutine then the program name must match the application IMS subroutine name.
- **Override Name:** Leave it blanks or fill in the same value as Program Name

- **Language:** the language the source program is written in. Choices are COBOL, PL/I or Natural.

- **Host Code Page:** values entered by the Administrator into the Admin Menu / Property File – **codepage.xml** that enable user to choose code page conversion to and from UTF8 are displayed here. The default is EBCDIC code page 37; CCSID 1140 is supported and is the Euro currency update of code page CCSID 37. In that code page, the "¤" (currency sign) character at code point 9F is replaced with the "€" (Euro sign) character.

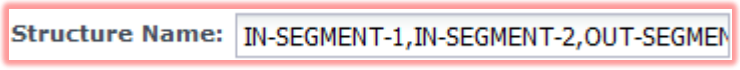
  Note: Code page values stored in codepage.xml and selected during IMPORT are validated when the service is executed. It is important to make sure the code page you are using during IMPORT is valid.

- **Enumerations:** If enumerations (viz. 88 level items in COBOL) needs to be imported or not. Choices are Include or Exclude

- **Environment:** the created program's environment. The environment is a custom property in SOLA and available environments will depend on your particular installation. Some examples of environments are "Test", "QA" and "Production".

- **Program Description:** a brief free-form description of the program.

- **Structure Name:** this is a comma separated list of all of the input, output and input/output(both) structures that will be used by the web service. These names must match the names of the structures in the program.



- **Class Name:** when you expose a program as a web service, its operations will be exposed as methods. Distributed systems classify methods as belonging to a class. Therefore, SOLA requires that you assign a class name to the program.

- **Dataset / Listing Name:** the input source. As mentioned previously, SOLA can import a commarea program from a compile listing (either saved or from the JES output queue) or from one or more copybooks. A compile listing is preferred because it allows SOLA to attempt to categorize the interface fields, saving you work during analysis.

- **IMS Transaction:** the transaction ID under which the IMS program will be executed. When the IMS Program Type field (see below) is "Main", this should be set to the transaction ID of the corresponding IMS program. When the IMS Program Type is set to "Subroutine" this should be set to the SOLA IMS Driver Transaction ID (SOLA is shipped with the Common Driver Transaction XML#IMCM). You can customize this by creating your own transaction that executes the IMS Driver Program XMLPC260.

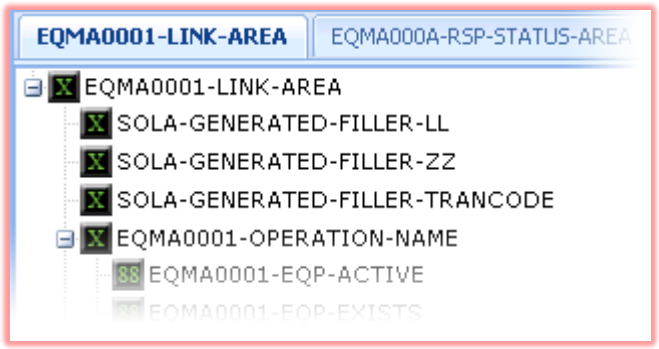
- **IMS Terminal:** this field allows you to specify a Terminal ID, if it is required by the IMS transaction. This is only required for applications that are coded to work on a specific Terminal.

- **IMS Natural Lib:** the Natural Library Name, containing the Natural Load Module.

- **IMS Program Type:** specify the program type, Main or Subroutine. For IMS Programs that accept input and output Segments and can run under their own transaction IDs, specify Main. For IMS Subroutines, which are invoked by COBOL programs within an IMS region, specify Subroutine (and use Subroutine Name as the "Program Name").

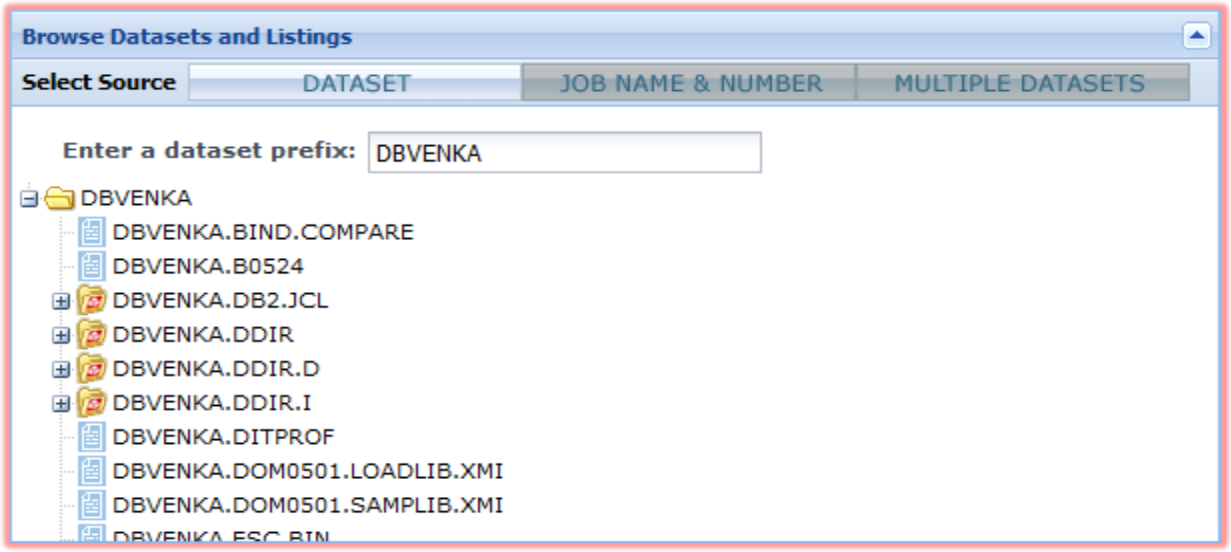
- **IMS LLZZ Prefix:** instructs SOLA to either generate or not generate the IMS LLZZ/Trancode prefix in the IMS segments being imported. This can be used when importing application structures that don't have the LLZZ/Trancode prefix. Choosing to generate will create extra LLZZ and Trancode(8 bytes) fields in the template

  You can see the generated prefixes in the image below:



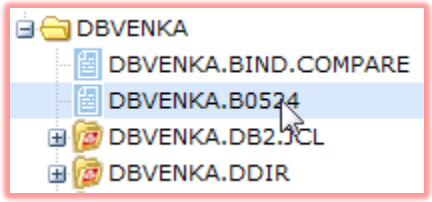
- **IMS Segement IdByLength:** The default processing of segments returned by application is to match them in the the same order as the output segments were specified in the Import screen. If you want to override this matching logic and identify the returned segments and match them to imported segment structures based on their lengths then override the default. This feature is useful when your application returns multiple segments having multiple structures and the order of segments returned is not fixed.

At the bottom of the Import panel is the **Browse Dataset and Listings** panel. This panel allows you to pick the input source from a list without having to manually enter it into the **Dataset/Listing Name** field.



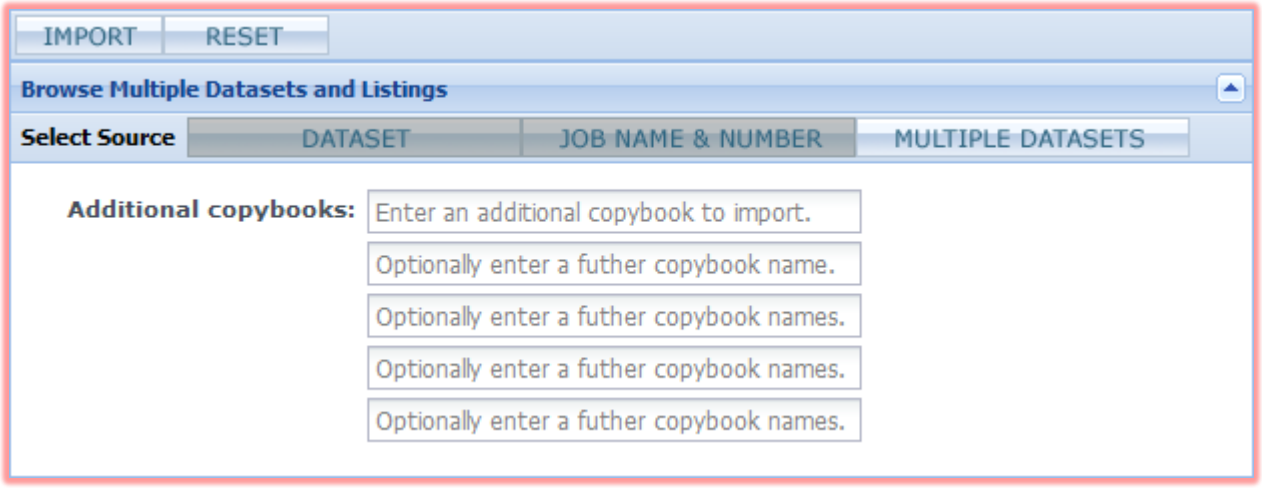
To use this panel, select from one of the three available source types by clicking on the appropriate button tab.

The Dataset option includes both saved compile listings and copybooks. You can change your default dataset prefix by entering a new value in the **Enter a dataset prefix:** field. Your default dataset prefix is a user-level custom property that can be set in your user properties (page 4).

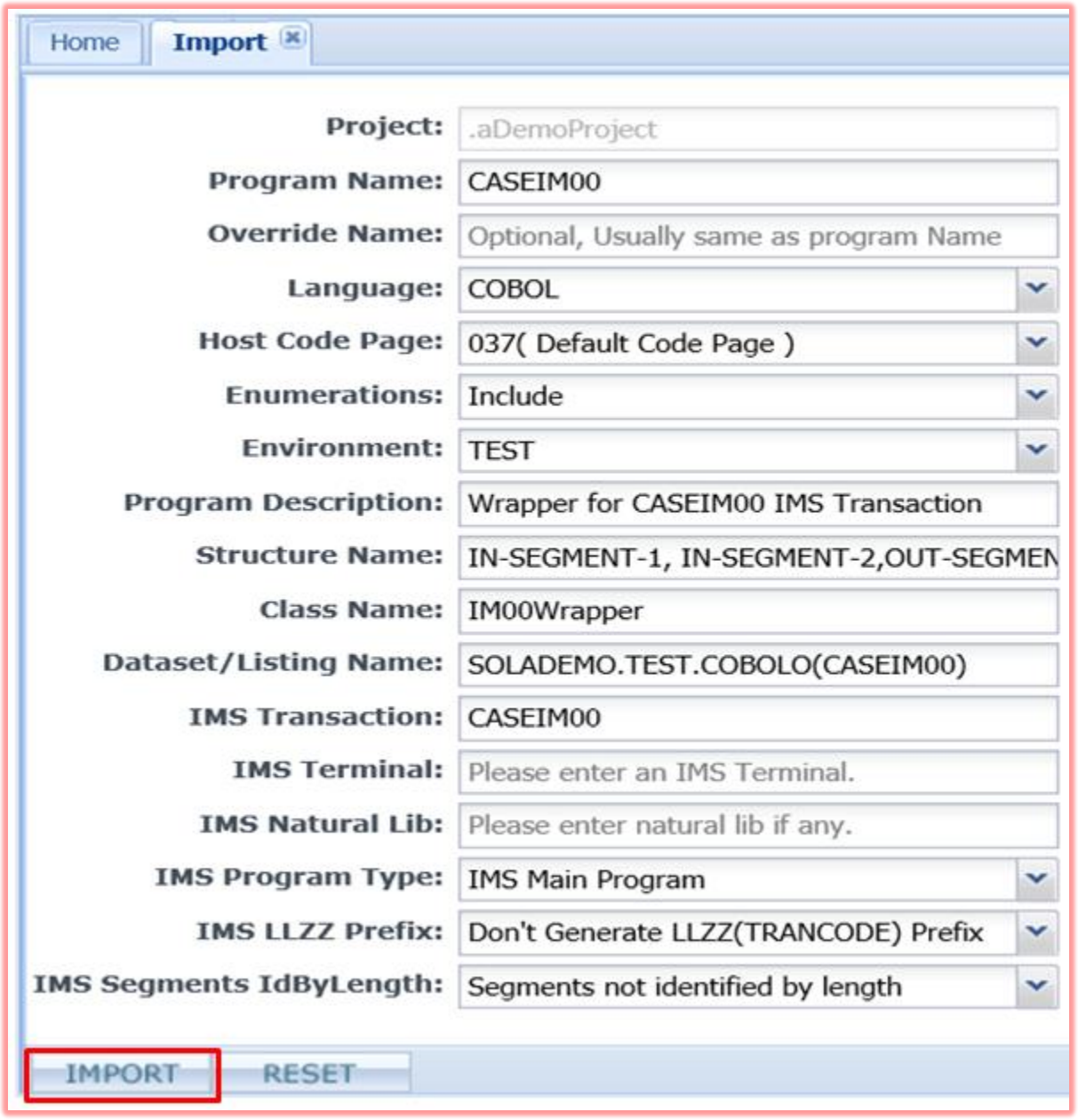


Once you have located the dataset or listing you want to import from, double click the dataset/ listing name to populate the **Dataset/Listing Name** field with your selection.

If you select **Multiple Datasets**, you will not be presented with a directory tree. Instead, you will be given five blank fields that you can use to specify up to five copybooks.



When you have filled in all required fields and are ready to import, click the **IMPORT** button.



Upon successful import, a confirmation message will be displayed. The newly created program will appear in the SOLA directory under the project you chose to import it into.

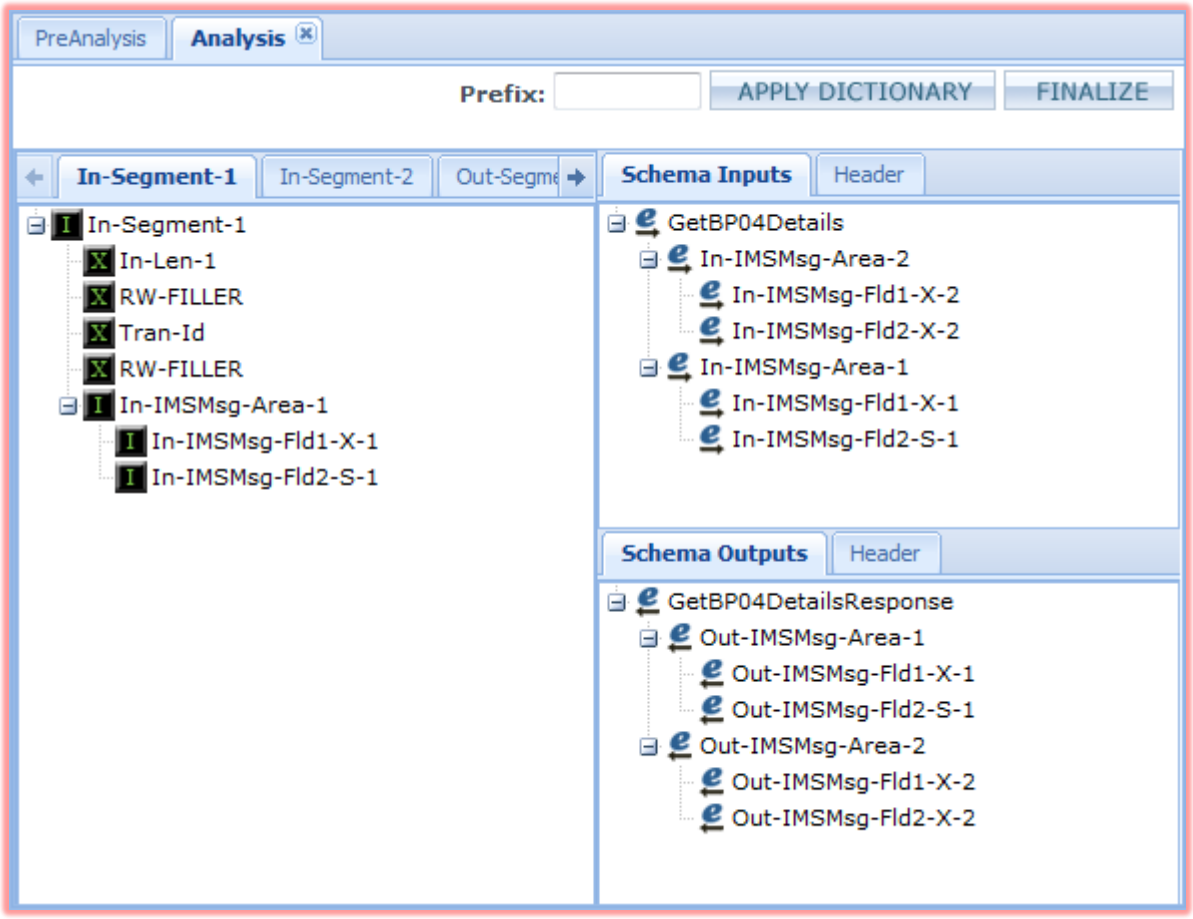
When importing IMS programs, the creation of methods is a separate step from the importing of the program. The following section will detail the creation of an IMS method.

Step 3 – Creating Methods in an IMS Program

Once an IMS program has been imported, you can create methods by isolating individual functions within the program. The IMS analyzer is almost identical to the commarea analyzer, with some differences that will be explained in this section. To understand how to use the IMS analyzer, you must first read the section on the bottom-up commarea analyzer (page 56), then return to this section.

Because IMS programs handle multiple segments, in SOLA they contain multiple interfaces (what would be called a "commarea" in a commarea program). Such an interface can be either input, output or input/output (both). When an IMS program is imported, all of the interfaces that will be used by the web service are identified, and all of the identified interfaces can be used during analysis.

The multiple interfaces appear as tabs in the legacy tree.



The tab name matches the name of the interface (for example, the COBOL 01 level) that you supplied during analysis.

SOLA IMS plugin has been enhanced to additionally generate SOLA IMS Override fields in the WSDL as part of soap:Header. To generate the optional header fields use the switch "**imsheader=true**" as follows:

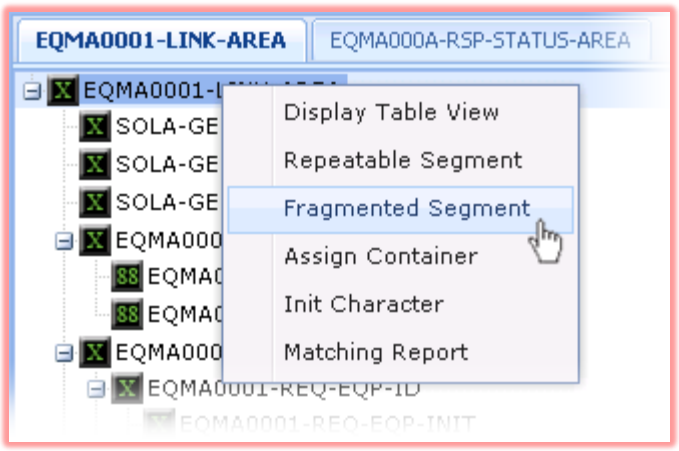
The WSDL will get generated with SOLA IMS Override fields (underlined in the example below):

Other than this one difference, the IMS analyzer is identical to the commarea analyzer.

Working with Fragmented Segments

IMS is capable of circumventing the 32K limit by fragmenting a structure into multiple 32K chunks. SOLA can handle programs with fragmented output segments, provided that only one application output structure is fragmented.

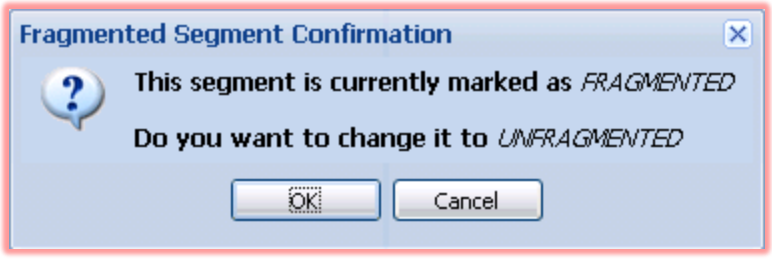
To specify that an output structure is fragmented, right click on the structure's 01 level during analysis and select **Fragmented Segment.**



A dialog box will appear asking you to confirm the change from unfragmented to fragmented. Click **OK** to continue.

The SOLA runtime will automatically combine the fragmented structure segments before constructing the soap response.

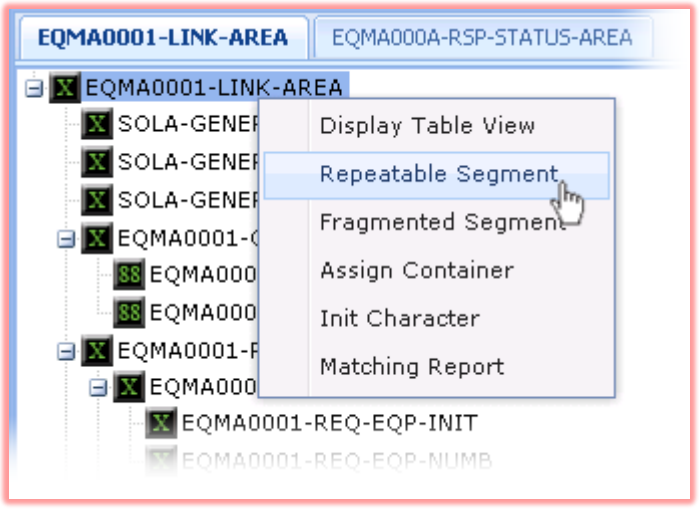
If you selected a structure that is already fragmented and then select **Fragmented Segment** from the right click menu, you will be presented with a dialog box that tells you the segment is currently fragmented.



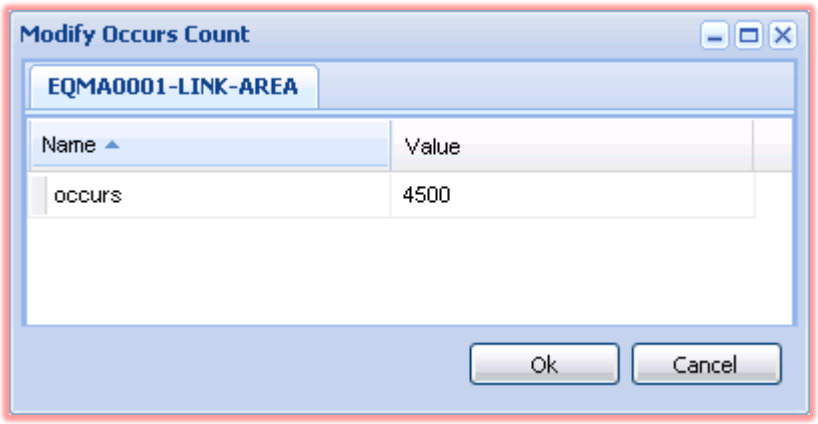
Working with Repeatable Segments

Repeatable segments are used to tell SOLA that some of the input/output data is to be mapped to the same segment. This is similar in function to an array and is used when large volumes of output data use the same format (e.g. customer name, DOB, address, etc.).

To mark a segment as repeatable, right click on the structure's 01 level during analysis and select **Repeatable Segment**. SOLA supports handling of repeatable segments in both input and output



A dialog box will appear asking you to specify an upper limit (you can set this as high as you want).



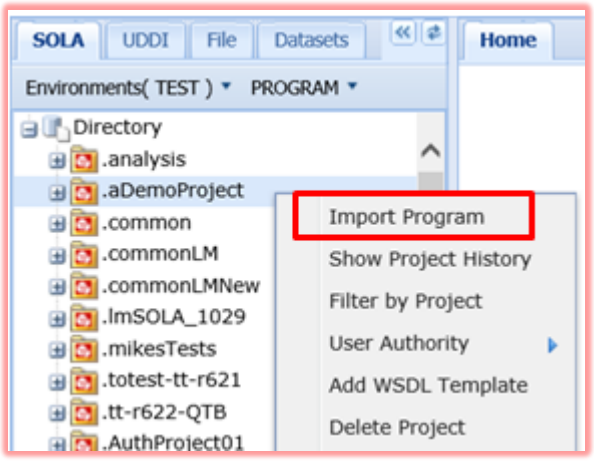
Creating an IMS Web Service- Top Down and Meet-in-the-Middle

Just as with commarea programs, it is possible to create a web service using a WSDL as a guideline, then create an IMS program around that web service (or alter an existing program), or work with both a WSDL and an existing program, merging the two. The process for doing so with an IMS program is nearly identical to the commarea process. The only difference is the multiple interfaces that IMS programs use during analysis, as discussed in the previous section (starting on page 146).

Step 1 – Mainframe Preparation

The only mainframe preparation required with the top-down and meet-in-the middle approach is for you to identify a PDS dataset for it to store the generated copybook (make sure that it is LRECL=80). You'll then write a program that incorporates that copybook, and you'll put it in a library that your IMS message processing region can access.

Step 2 – Importing the WSDL

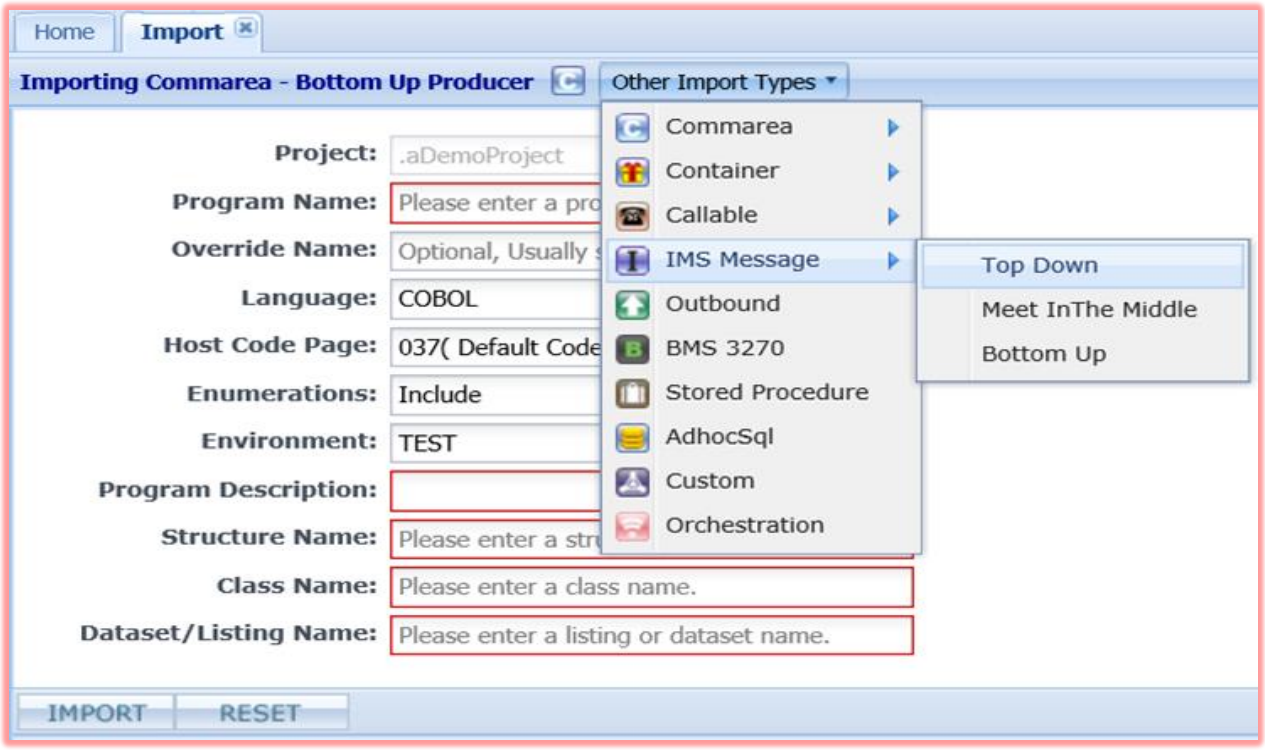


To get to the top down import panel, select the project you wish to import to and right-click it. From the pop-up menu, select **Import**.

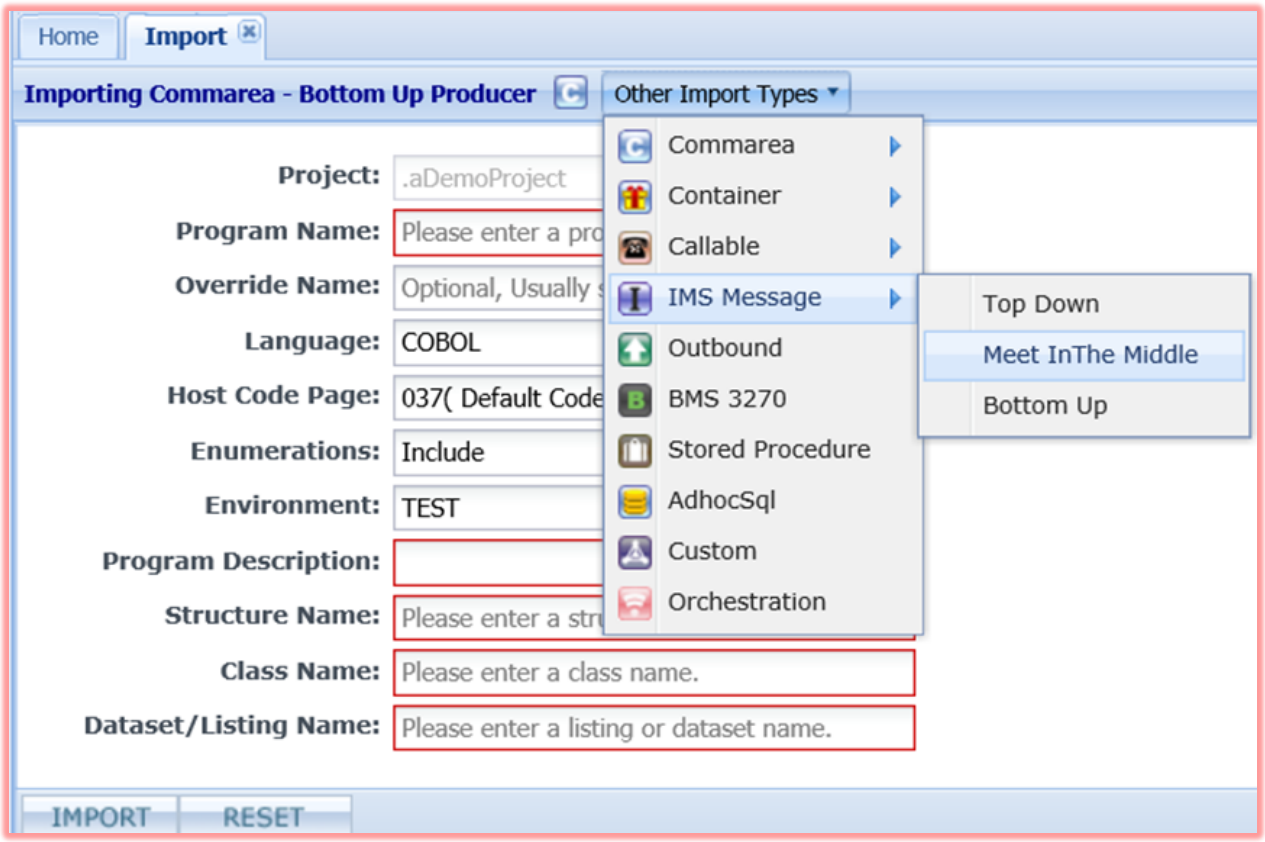
After you select **Import**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

The default program type is commarea bottom-up, so use the **Other Import Types** menu to select either IMS Message **Top Down** or **Meet InThe Middle.**

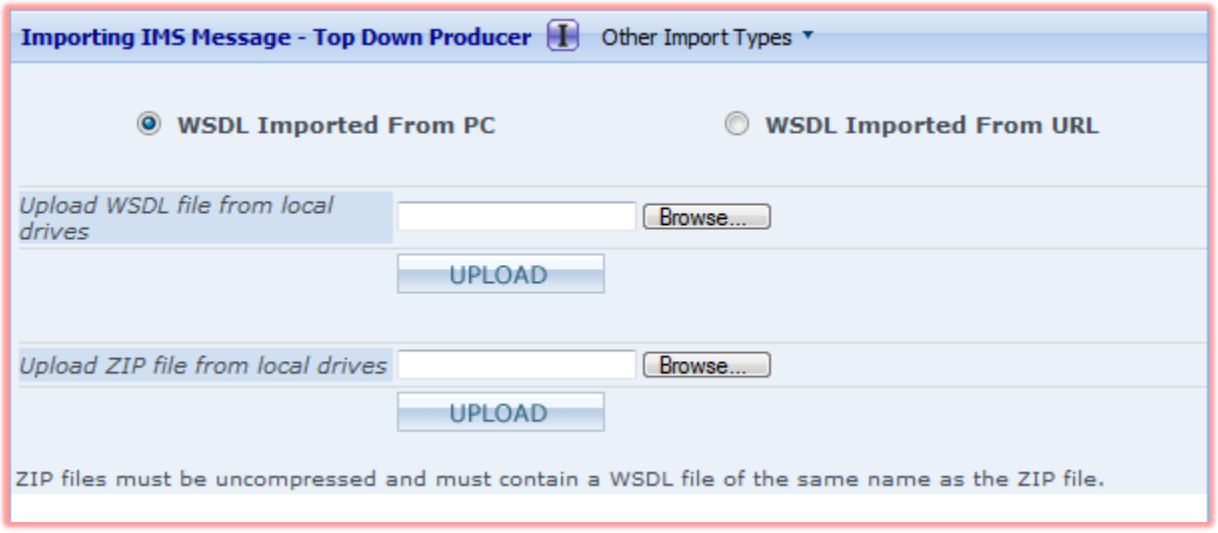
Top Down:



Meet InThe Middle:



After clicking the **IMPORT** button The Import panel will change to display the WSDL import panel.



This panel and the associated Import from URL panel are identical to those used in top down and meet-in-the-middle commarea import (page 76 and page 82, respectively).

Note the **Host Code Page** selection options (*this is described further in the SOLA Administration Users Guide and in the 'Importing a Commarea Program' and 'Admin Menu' button sections of this guide*.)

Step 3 – Analyzing the WSDL to Create the Copybook

IMS top down and meet-in-the-middle analysis is identical to its commarea counterparts, with the exception of multiple interfaces and fragmented and repeatable segments described in the IMS bottom-up section (page 128).

Using SOLA Developer – BMS 3270

For many years, the 3270 terminal was the principal method of communicating with CICS transactions. Despite years of investment in alternatives, billions of these transactions continue to be run every day by companies across the globe. In fact, in many companies, the 3270 transaction is still the most common way to access CICS transactions.

Although the physical 3270 terminal is long gone, the 3270 emulator remains to provide a PC based alternative or a "screen-scraping" solution through the HLLAPI.

The reason that so many 3270 applications remain is because they are extremely efficient, highly reliable and easy to operate. They have, however, proven very difficult to replace. This is principally because the CICS 3270 "pseudo-conversational" programming model is very difficult to port to other environments. In a pseudo-conversational transaction the 3270 operator executes multiple iterations of 3270 transactions to perform a single business transaction. When developing a screen-scraping solution, the programmer has to understand the countless ways that 3270 transactions are constructed and he/she has to build a complex driver to emulate the myriad operator interactions.

How SOLA Creates Web Services from BMS 3270 Transactions

SOLA has eliminated the complexity involved in making 3270 transactions available to the web. SOLA attacks the complex 3270 problem by focusing on the "business transaction", not the individual pseudo-conversational transactions addressed by other approaches. SOLA doesn't use screen scraping, instead it runs natively in CICS and interfaces with the CICS supplied 3270 Bridge.

When creating web services from a series of 3270 transactions, SOLA combines a complex chain of pseudo-conversational interactions into a single request/response operation, or "use case". For example, think of the interaction between man and machine when you use an ATM to withdraw cash from your bank account. There are many possible interactions when interfacing with an ATM, but withdrawing money from your checking account is one particular use case. Another use case may be depositing money into your savings account. SOLA allows you to expose each use case as a single web service operation.

To do this, SOLA provides an Analyzer for 3270 transactions. When creating a web service from a 3270 transaction, you run your use case through the Analyzer, teaching SOLA how to run it. Once you've successfully taught the Analyzer how to run the use case, the Analyzer creates WSDL, meta-data and a test harness and documents the transaction in the SOLA UDDI directory.

Before you can use SOLA to analyze a transaction, you need to understand how the transaction runs; the screen flow, the inputs and outputs, etc. One way to accomplish this is to run through the transactions you want to expose, screen by screen. Once you understand how the transaction is run, you can then use the SOLA Analyzer to expose it as a web service.

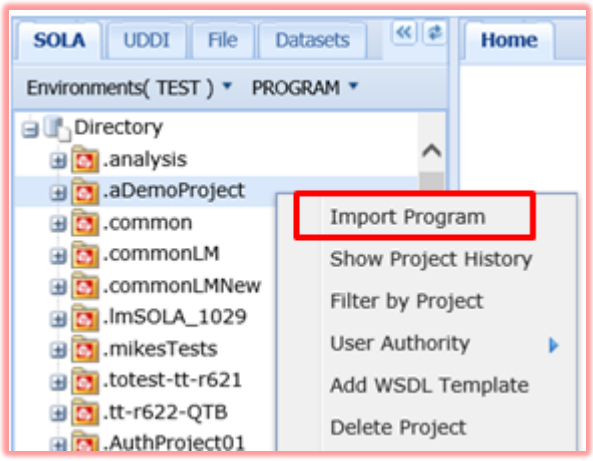
Creating a Web Service from a Simple BMS3270 Use Case

This section will describe the steps necessary to create a web service from a series of BMS3270 transactions.

Step 1 – Mainframe Preparation

Before launching the SOLA developer, you should run through the use cases that you plan to import. It is a good idea to not only run through each use case to its conclusion, but also to experiment with faulty inputs and other ways to generate errors to see how the program responds. If you are very familiar with the 3270 program, this step is not required.

Step 2 – Importing and Analyzing the Use Cases

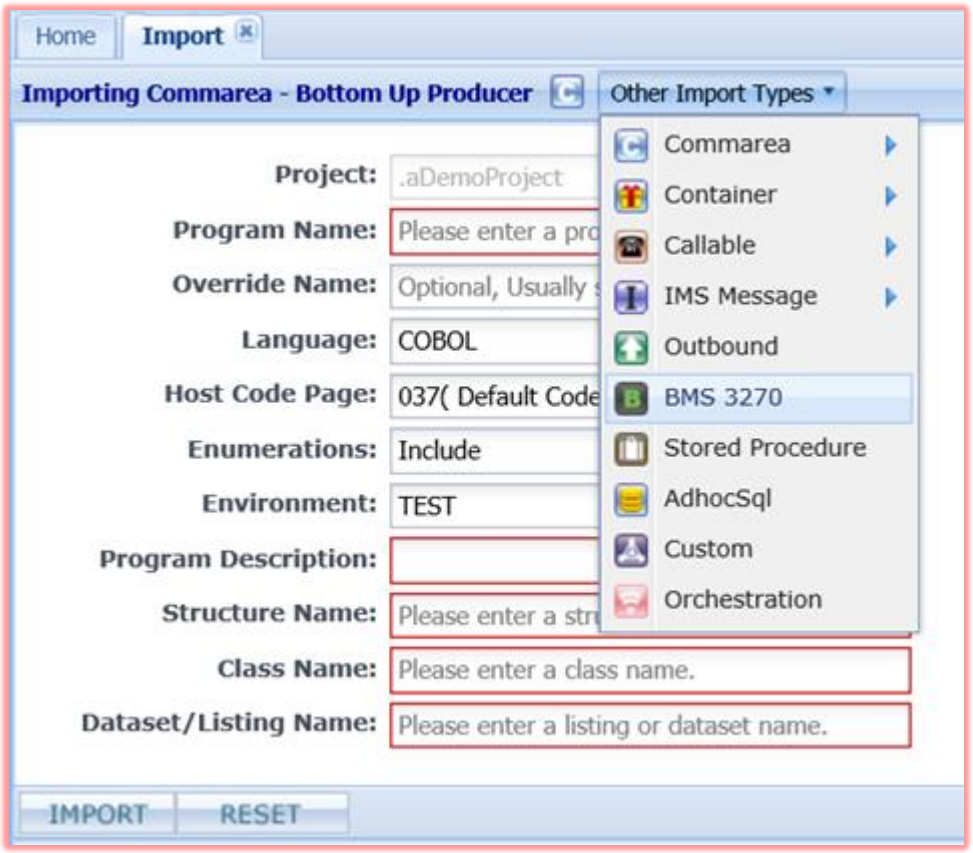


Select the project you wish to import to and right-click it. From the pop-up menu, select **Import Program**. If you wish to import the program to a new project, first follow the steps for creating a new project on page 22.

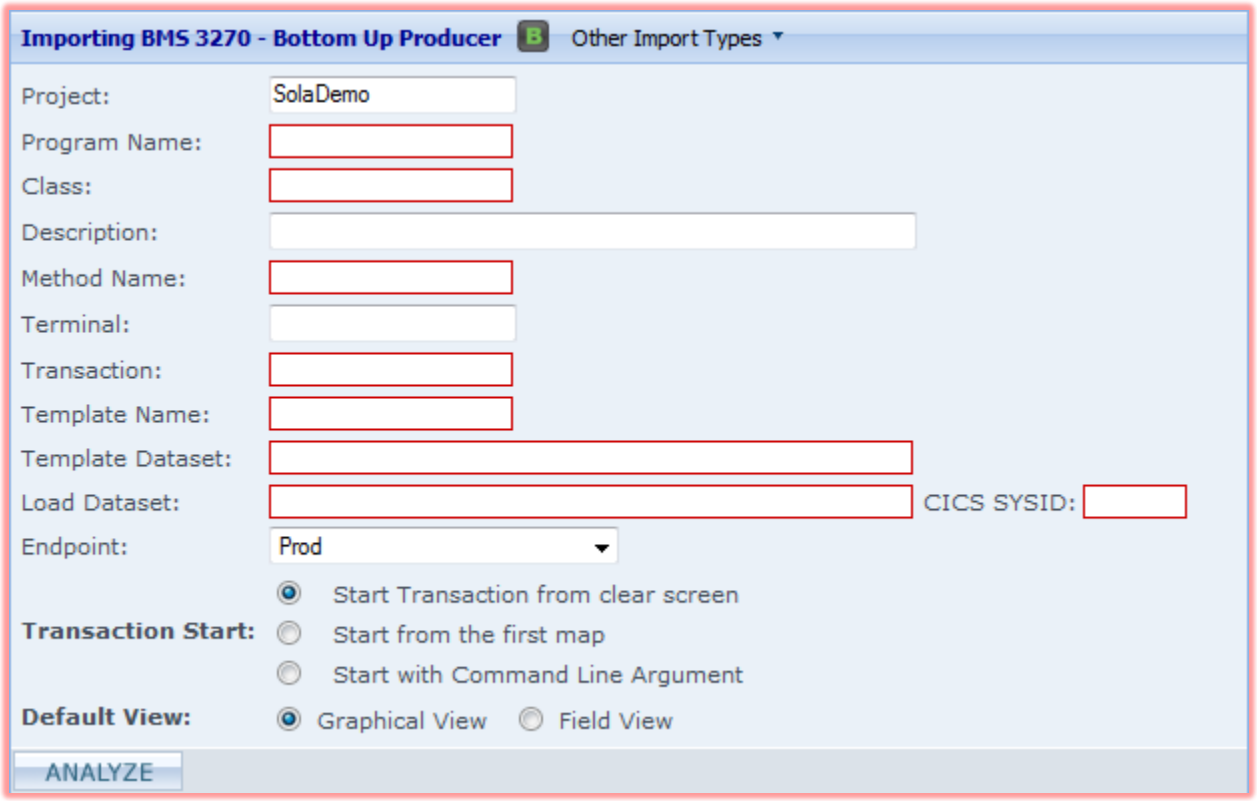
When creating web services from 3270 use cases, importing the parent program is done in conjunction with creating a method. Unlike commarea programs, a 3270 program cannot exist on its own (without methods). The program is nothing more than a container for methods.

After you select **Import Program**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

The default program type is commarea, so use the **Other Import Types** menu to select BMS3270.



The Import panel will change to reflect the selected program type.



The Import panel consists of a series of fields used to provide information about the source program and the destination SOLA program that will be created.

- **Program Name:** this is the name that the imported program will be stored under (the use cases will be methods of this program). This doesn't have to be the actual name of a program associated with the transactions, but it should be meaningful to the creator of the web service.
- **Class:** when you expose this program's use cases as a web service, each use case will be stored as a method. Distributed systems classify methods as belonging to a class. Therefore, SOLA requires that you assign a class name to the program.
- **Description:** the description of the class for documentation purposes. This is pre-filled from project-level properties but can be changed.
- **Method:** each use case must have a unique method name.
- **Terminal:** Terminal is only used if there is a terminal dependency for the program – for example if you need to have signed on to a security system before you can execute the transaction. In that case you would sign on using your 3270 emulator and then enter the terminal ID where you've signed on in this field.
- **Transaction:** Transaction is the 4 character transaction ID that you enter from a blank screen to invoke the use case.

- **Template Name:** this field tells SOLA what you want to call the template (run-time metadata) that will be created when the use case is analyzed. The template tells SOLA how to facilitate communications between a legacy application and a distributed client or server. A template is an Assembler Data Only Load Module.

- **Template Dataset & Load Dataset:** these fields are used to tell SOLA where you want the template source and the compiled and link-edited template to be stored. The source of the template will be stored as a member in the Partitioned Data Set (PDS) named in the **Template Dataset** field. The SOLA Analyzer will automatically assemble and link-edit the template into the Load Library specified in the **Load Dataset** field.

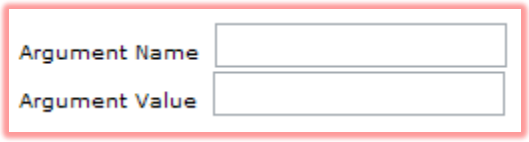
- **CICS SYSID:** the 4 character CICS SYSID where the transaction runs.
- **Endpoint:** the region in which SOLA is running.

In addition to the fields, you have several options for how to start the transaction.

- **Start Transaction from clear screen:** Choose this option to have the web service start from a clear screen. This is the default option.

- **Start from the first map:** Choose this option to have the web service start from the transaction's first map.

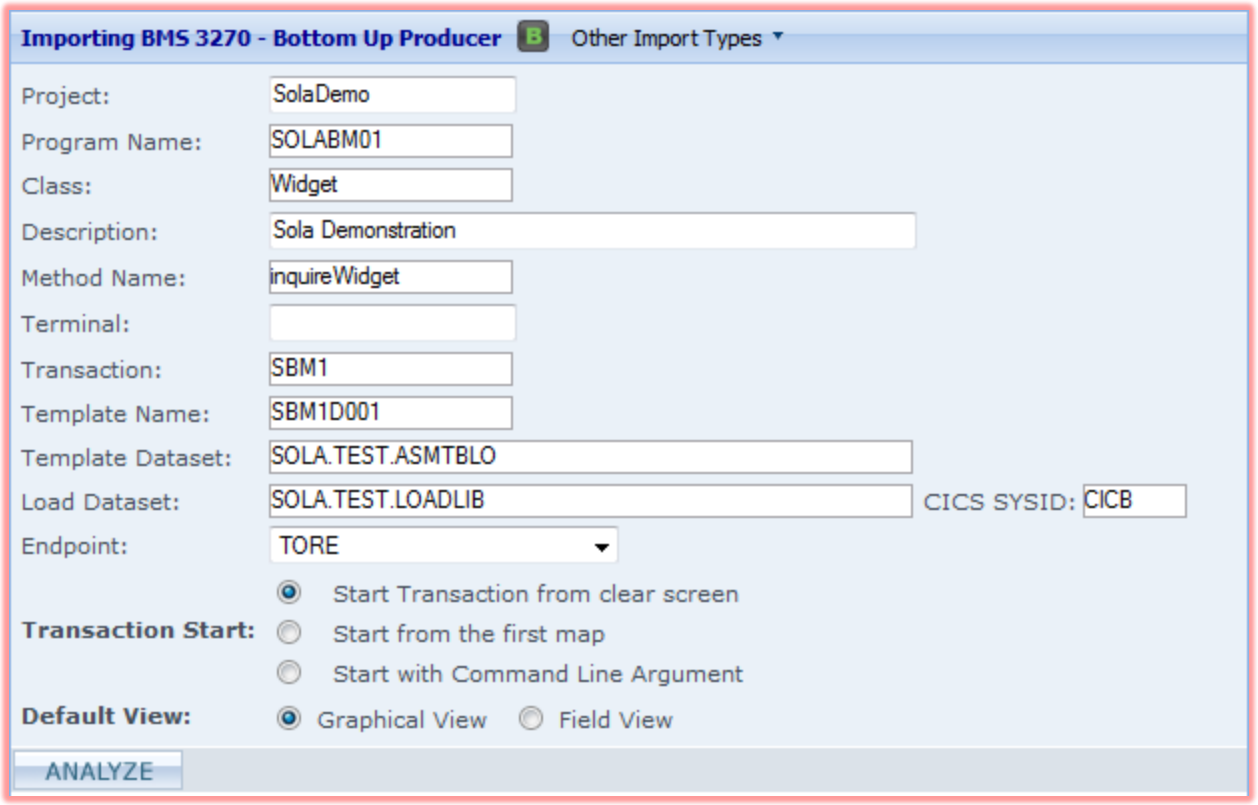
- **Start with Command Line Argument:** Choose this option if you want the transaction to start with a command line argument. Selecting this button displays two additional fields. Use these fields to enter the command line argument and its value. To use multiple arguments, string them together in sequence. If you enter an argument and a value, the transaction will be executed (during analysis) using the supplied information.



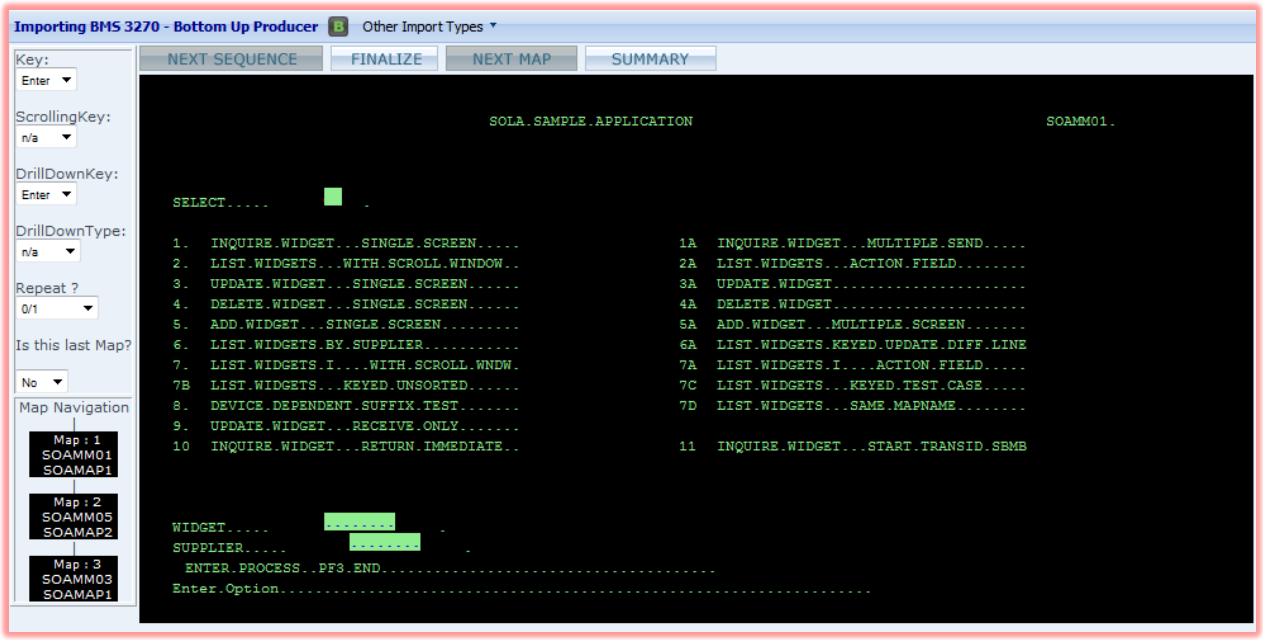
- **Graphical View:** chose this option to launch the 3270 Analyzer with the default graphical view.

- **Field View:** click (select) this button to launch the 3270 Analyzer with the optional field view.

When you have filled in all required fields and are ready to import the program and begin analyzing the first use case (you cannot Import a 3270 program without creating at least one method), click the **Analyze** button.



After you click **ANALYZE**, the Analysis panel will display the BMS3270 Analyzer.



In the default view, the 3270 Analyzer is a graphical representation of a green screen with some differences related to functionality: all empty spaces in all fields are painted with dot characters (e.g. ……), which aids in locating hidden fields. In recreating the green screen, SOLA

has captured not only the look of the original program but its functionality as well. You can actually run through the entire 3270 program within the Analyzer. To create a method from a use case, however, you should only run through the specific use case you are creating.

For example, the program shown in the illustration above has 20 options, and each option may lead to other options. This program may represent hundreds of use cases. An example of a specific use case would be selecting option 1 (information about a specific widget), entering the widget number in the WIDGET field and advancing to the next map with the Enter key. The next map should contain information about the widget number provided in the first map. This is a simple but complete use case, and it can be likened to a customer requesting information about a specific account. You can build a modern web based interface for the customer to interact with, and by exposing use cases as web services; the customer can be invoking the legacy program and data without realizing it.

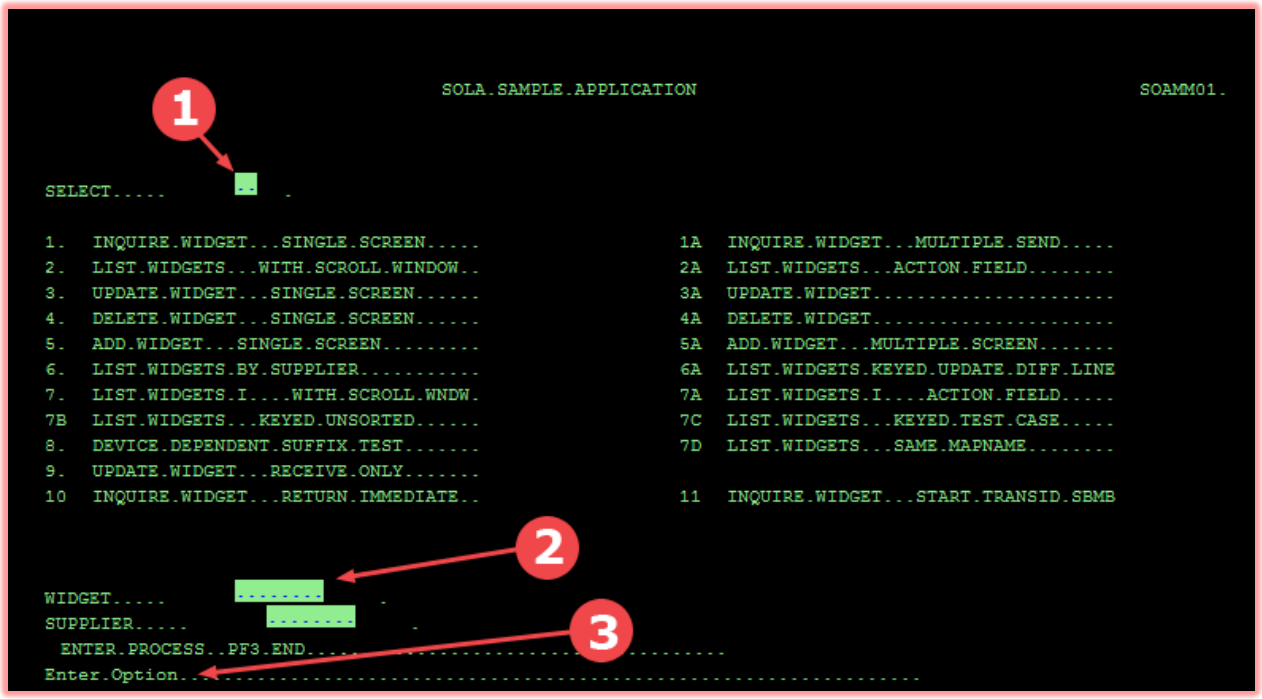
Creating a Web Service

When creating a web service using the SOLA Analyzer, you are essentially doing the same thing as creating a web service from a commarea program. You are creating a WSDL, which describes the interface to the program; the input and output fields. As complex as a 3270 use case may seem, it is really nothing more than a group of inputs and outputs. The inputs are the fields the green screen program requires as well as the keys(button presses) it needs to advance maps (e.g. Enter key to advance maps, PF1 to drill down, etc.). The outputs are the data fields that the program returns.

The 3270 Analyzer provides a very simple interface for describing both input and output fields. This interface allows for a wide range of field types, from simple input or output to error message or end marker.

In describing how to create a web service from a use case, we will use the simple use case mentioned earlier in this section (requesting data about a specific widget). Creating this simple web service will allow you to understand how the Analyzer works and give you the skills you need to create more complex web services.

The first step in creating our simple web service is to identify the input fields required for the use case.



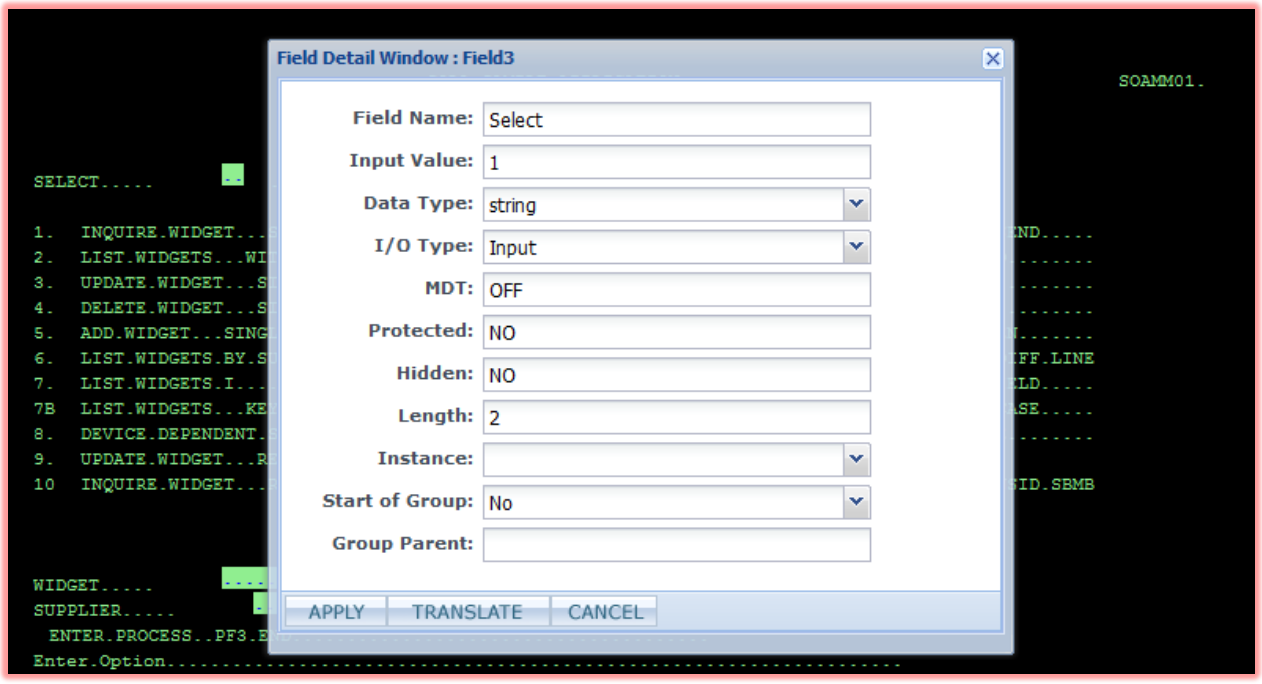
1 This is the input field that you must use to select an option from the green screen menu. For this particular use case, we will be choosing option 1.

2 This is the input field that you must use to enter a widget number to inquire on.

3 If you ran through the transaction in the green screen terminal, you would see that entering an invalid widget number in field 2 results in an error message displayed in this field. When creating web services, you can specify the number of times a certain map is displayed. In our use case, this map should only be displayed once. If it is displayed twice, it is because of an error (such as entering an invalid widget). SOLA will know that an error has occurred, because a map that should only have been displayed once has now been displayed twice, and will throw a SOAP fault. If you wanted to capture the legacy error message in the SOAP fault, you could describe this field in the WSDL as an error message field. SOLA would then use the text contained in this field in the SOAP fault it throws.

Now that we know what fields we need on this first map, it's time to describe them for inclusion in the WSDL.

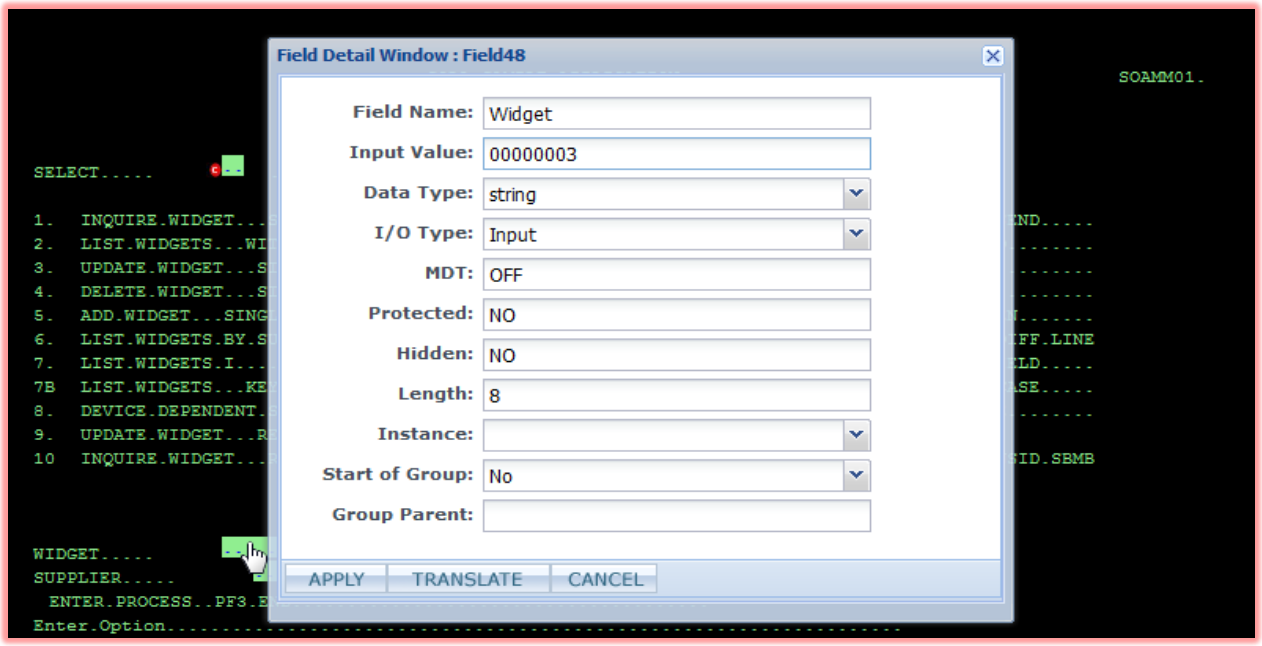
To describe a field, right click it. SOLA has identified the input and output fields on the green screen representation, so clicking anywhere on an output field or within the green box on an input field will bring up the field menu.



When configuring fields, you must change the field name, or else SOLA will ignore the field while creating the web service. In this instance, we have assigned a default value of 1 to the field, which we've named "Select". In the image above, the IO type is set to "Input". This means that while SOLA will run the transaction with the value we supplied (in this case 1), the WSDL would indicate that this is a user supplied value. Since passing a value of 1 is necessary for this use case, we will need to set the IO type to "AlwaysDefault", so that the web service will always supply a value 1 for this field when running the legacy program.

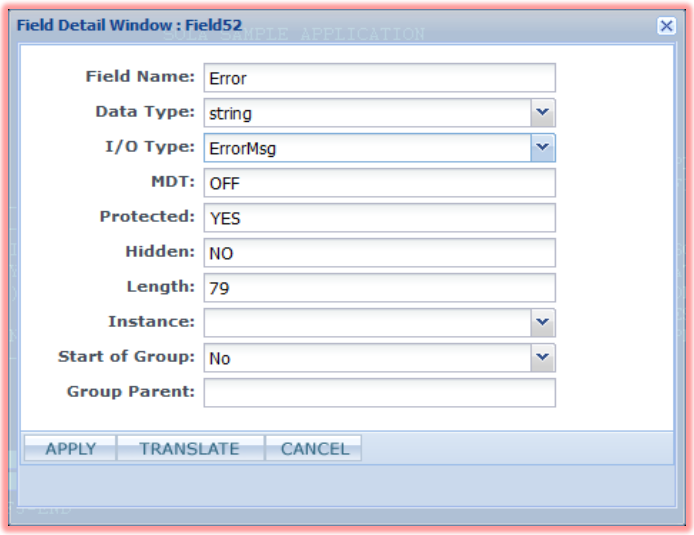
Once you have configured the field, click **OK**. The Analyzer will display an icon next to the field to indicate that it has been configured.

It is now time to configure field 2 .



Notice that we provided a value to this input field. When the web service is implemented, however, the user can provide any value that he or she wishes. Since we did not mark this field "AlwaysDefault" as we did for the previous field, this field will appear in the WSDL as an input field and the web service will accept inputs from the user. Why then do we need to provide a value when creating the web service? The transaction needs an input (in this case a valid widget number) to advance to the next map. SOLA must learn how to run the transaction, and the transaction must have inputs to run. This is why SOLA makes a distinction between regular input fields and "AlwaysDefault" fields. While analyzing this use case, we can provide any valid widget number.

Configuring field 3 is optional (the web service can just throw a generic SOAP fault), but not advisable.



Declaring a field's I/O type as an "ErrorMsg" means that SOLA will take the text of that field as a SOAP fault should it encounter a fault condition. That fault condition, however, has nothing to do with this field; the presence of an error message in a field declared an "ErrorMsg" field will not generate a SOAP fault. The Analyzer must be taught when to throw a fault. As mentioned previously, one example of this is declaring that a map must only appear once. If that map appears twice, then that means something has gone wrong and SOLA will throw a SOAP fault. If there's a field declared "ErrorMsg" on the map that caused the fault, the text in that field will be included in the SOAP fault. If there isn't, SOLA will throw a

generic SOAP fault. The absence or presence of an error message field, therefore, affects only the contents of the SOAP fault.

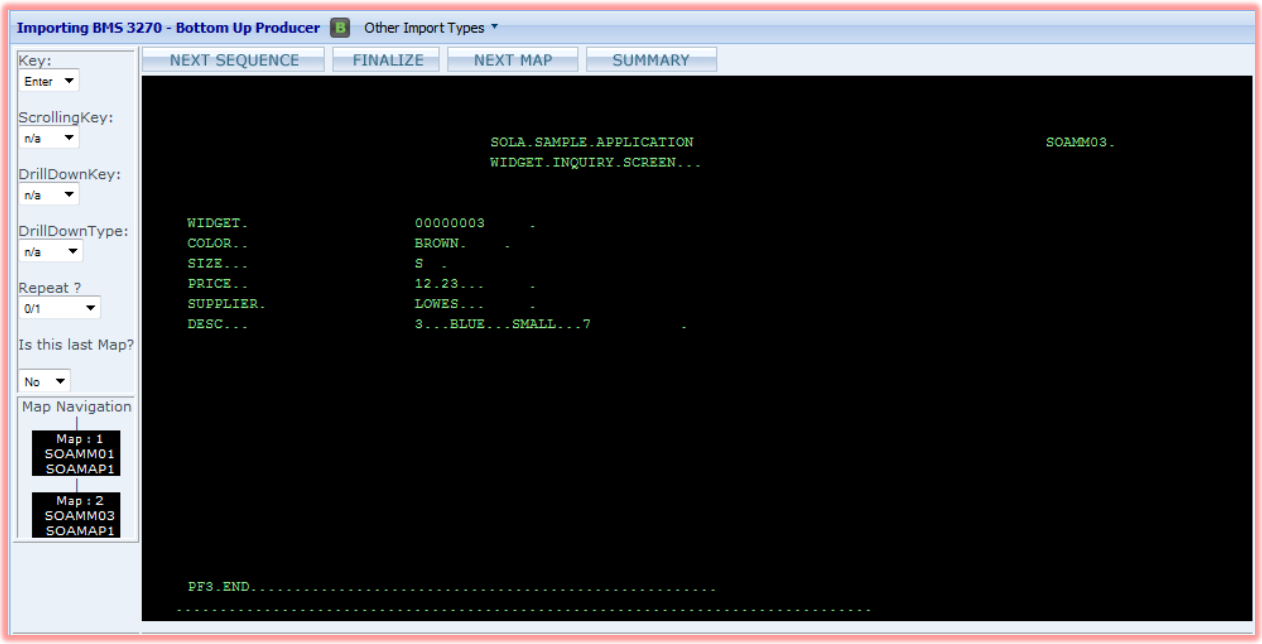
Now that we have fully configured the first map, it's time to set some map navigation options and advance to the next map.

1 The **Key** value is the keyboard key that needs to be pressed to advance to the next map. In 3270 applications, this is usually the enter key. The value in this menu has to match what is required by the 3270 application. Notice that on the bottom left of the green screen simulation is a key legend that tells users what keys to press. It indicates "ENTER=PROCESS", telling us that the Enter key processes our request. Therefore, in our use case, we should leave the default value of "Enter".

2 This value dictates how many times the map is allowed to repeat before a SOAP fault is generated. The default value is "0/1", which indicates that the map can only appear once (does not repeat). Options range from 0/1 to 4. There is also an unlimited value. If **Repeat?** is set to "unlimited", you must specify an end marker or the web service will enter an endless loop (information on end markers and other Analyzer functions and settings appears later in this section). Since our use case does not call for this map to be repeated, we should leave it at "0/1".

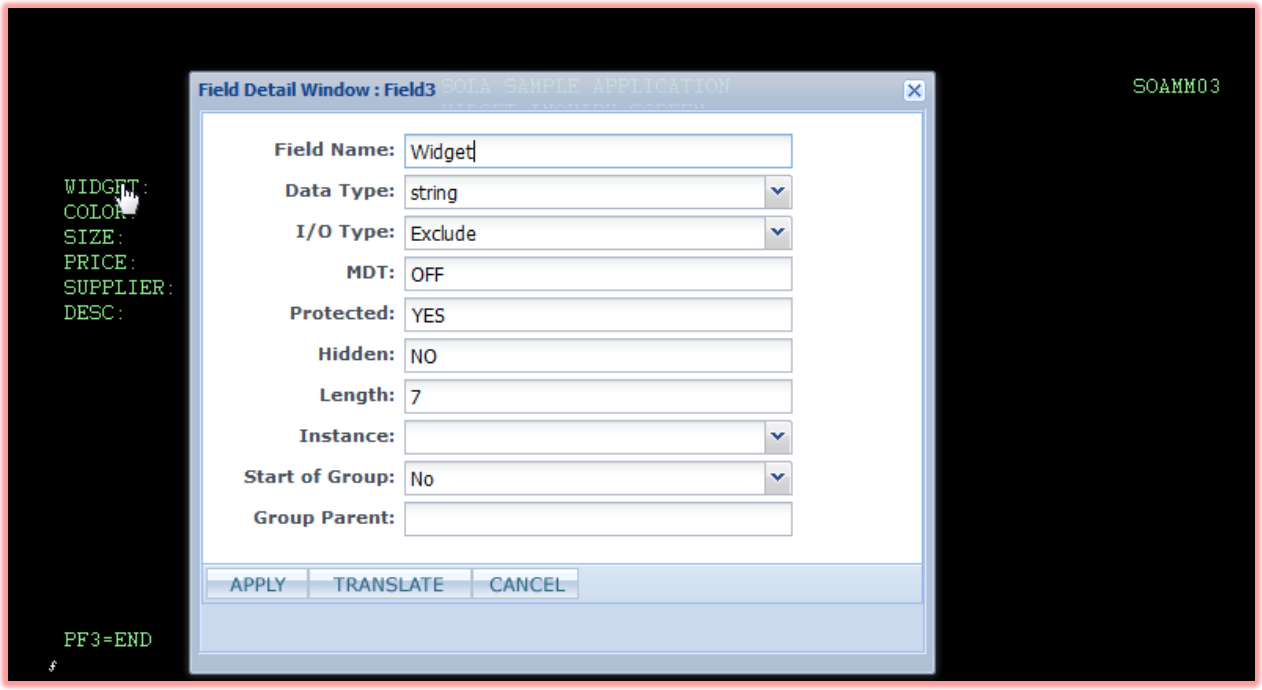
3 This value tells SOLA whether this is the last map in the use case. This setting is important, as it is one of the ways that SOLA knows the use case has come to its conclusion. In our use case, this value should be set to "No" for this map.

When you are ready to advance to the next map, you can click the **NEXT MAP** button.



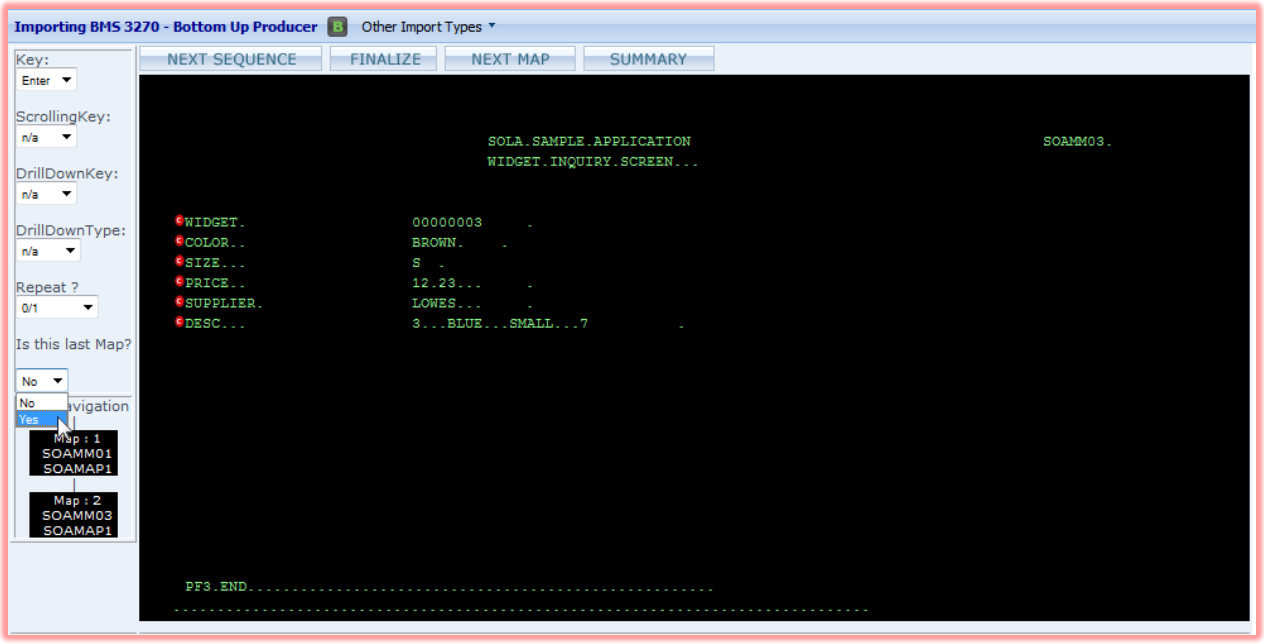
As requested, the 3270 application has provided information about the widget we specified. Now we must describe the output fields we are interested in for the WSDL and set an end condition so the web service knows when to terminate.

We are interested in all of the output fields that pertain to the widget, so right click on each of them and configure them.

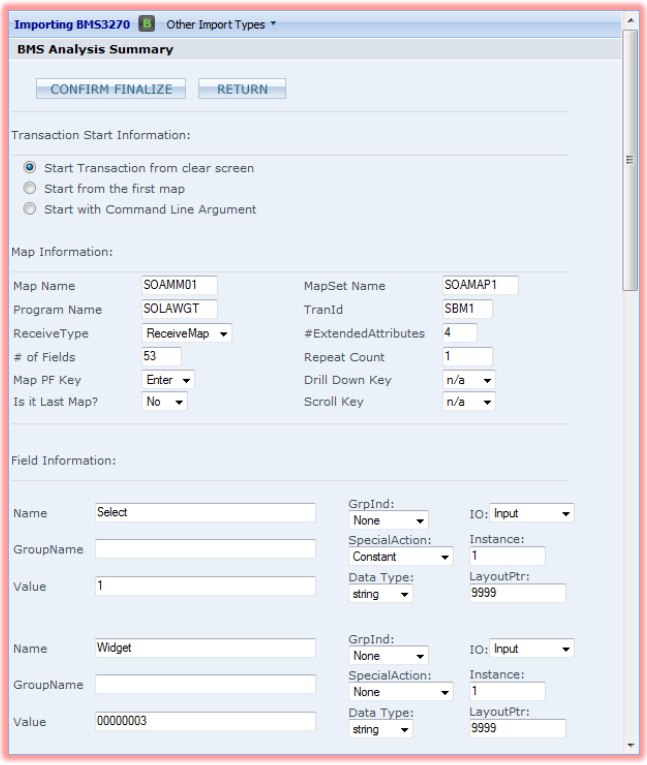


Since we're describing simple output fields, the only thing we have to do is change the field name.

Once you have named all of the output fields, the only thing left to do is to set an end condition. Since this is the last map in our use case, you can do that by changing the value of **Is this the last Map?** to "Yes".



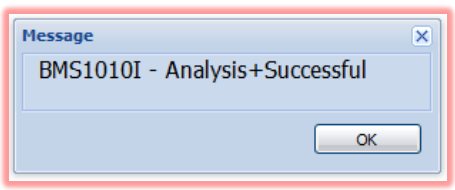
When you are finished, click FINALIZE to continue.



You will be presented with a summary panel that contains all the map and field information you configured during analysis. If there were any errors in your analysis, such as not specifying the last map or an end marker, this panel would also contain warning messages describing the problem.

You can browse this panel to make sure all of the maps and fields are configured correctly. If you find something you don't like, you don't have to go back to analysis, you can change it right here in the summary panel.

When you are certain everything is the way you want it, click the CONFIRM FINALIZE button. A confirmation message will be displayed.

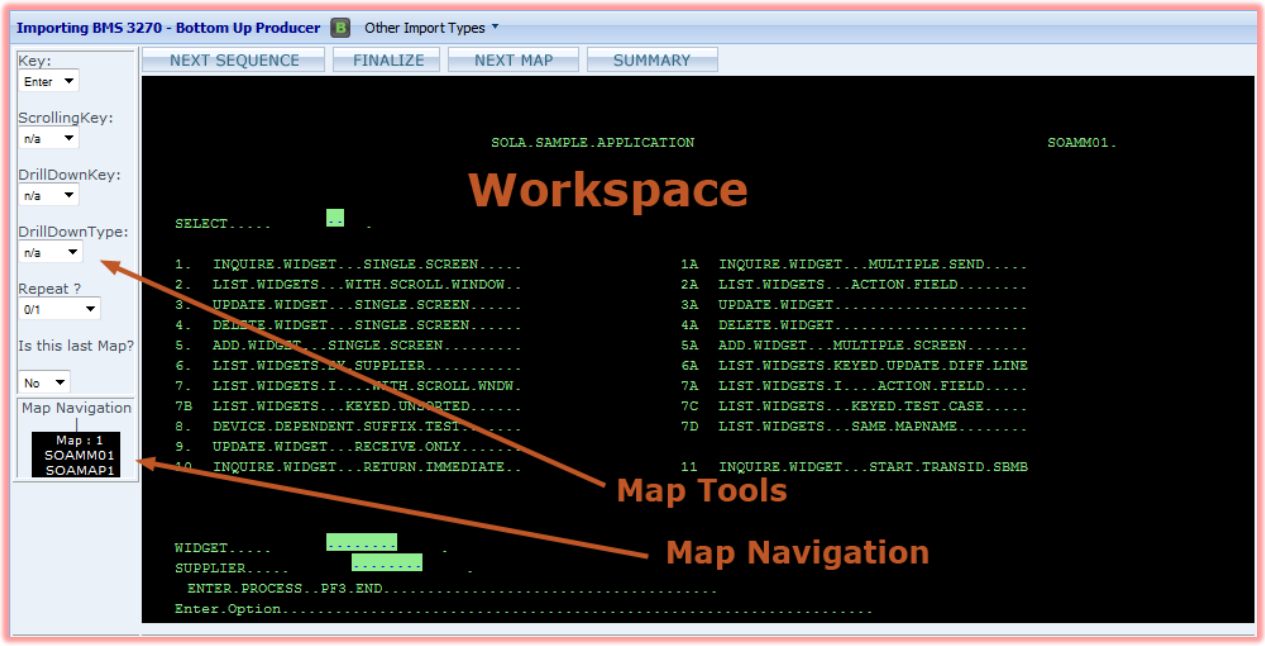


BMS3270 Analyzer Reference

This section will provide detailed information about the BMS3270 Analyzer. It is strongly recommended that you read this section in its entirety at least once before tackling any major projects using the Analyzer.

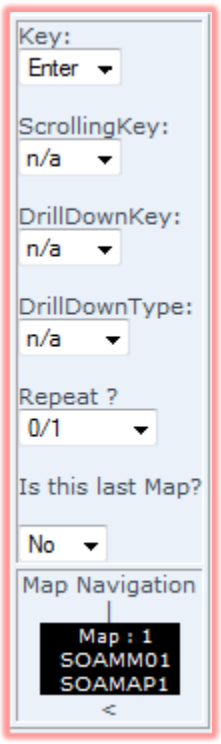
As SOLA is intended to be used by mainframe programmers, the BMS3270 Analyzer and this documentation are both designed to be easy to use by people with a certain degree of mainframe knowledge. If you are not a mainframe programmer, you may find some portions of this document to be intimidating or overwhelming. If this is so, then you should realize that SOLA was designed to be easy to use by everyone, not just mainframe programmers. Specialized mainframe knowledge and/or skills are not required to make the most of the BMS3270 Analyzer (though they do help). Anyone can use the Analyzer to expose complex series of transactions as web services, all it takes is a step by step approach, starting with simple use cases and slowly working up to more complex ones. The Analyzer is, underneath the surface, extremely intuitive and simple to use.

The BMS3270 Analyzer is divided into four sections; the workspace, the button bar, the map tools and the map navigation controls.



Map Tools

This tool bar is used to configure map navigation options such as key assignments, map repeat settings and more. This tool bar is also used to control the view in the BMS3270 Analyzer workspace.



Key: this is the keyboard key the user needs to press after making a selection. The default value is Enter, other options are PF1 – PF24 (function keys).

You can select a key (and simultaneously advance to the next map) by pressing either the Enter key or one of the 24 function keys (Shift + function key for PF13-24) when you are ready to advance to the next map.

NOTE: *Function key shortcuts, such as F5 to refresh or F1 to open Windows Help have been disabled in the BMS3270 Analyzer.*

Scrolling Key: specifies which key is used to scroll the screen (if the screen is scrollable). The default value is n/a (not applicable), other options are Enter and PF1 – PF24 (function keys).

Drill Down Key: specifies which key is used to make selections (drill-down) on the screen. The default value is n/a (not applicable), other options are Enter and PF1 – PF24 (function keys).

Drill Down Type: specifies the drill down type. The default value is n/a (not applicable), other options are ALL or Key.

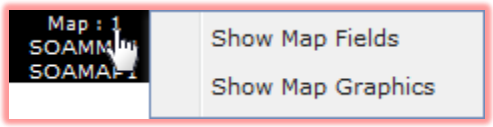
Repeat: indicates how many times the screen should repeat before an error message is generated. The default value is 0/1 (appears only once). Other options are 2, 3, 4 and Unlimited.

Is this last Map?: indicates whether this is the last screen in the transactions. Options are YES and NO.

Map Navigation Controls

This field shows a navigation map of all the screens involved in the transaction up to the current screen (displayed in the work area).

Clicking on a map from the main Analyzer screen reveals the Map menu.

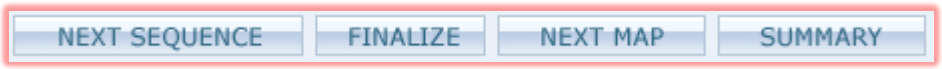


The options in the Map menu apply to the screen from which the menu was accessed, not the screen that is currently displayed in the work area (unless they are one and the same). Therefore, selecting a view option for the menu may change which screen is being displayed in the work area.
mode.

Show Map Fields: selecting this option will display the selected screen in the field view mode.

Show Map Graphics: selecting this option will display the selected screen in the graphics view

Button Bar



- **Next Sequence:** click this button to restart the transaction with a new set of maps (e.g., if your transaction takes two different sets of maps depending on type account number on the first map then you need to teach SOLA each sequence separately by entering two different account number for these two different sets).
- **Finalize:** click this button to complete and save the analysis.
- **Next Map:** Click this button to proceed to the next BMS 3270 screen in the transaction. You will not be able to proceed unless you have satisfied the requirements of the current screen (input required values, etc.). If NextMap is clicked while editing a method, SOLA will execute your transaction in real time.
- **Summary:** click this button to go to the BMS Analysis Summary panel.

Working with the Graphics View

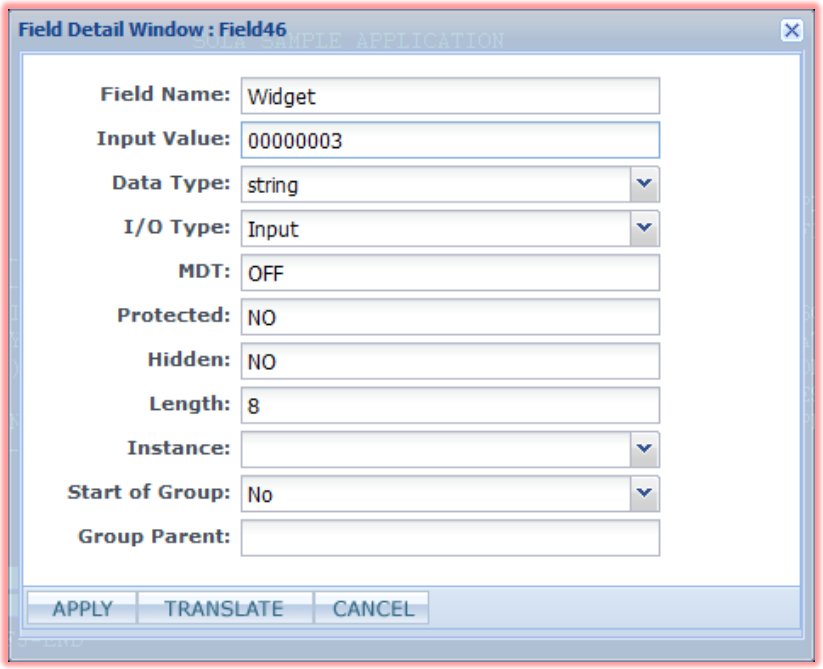
The Graphics view displays each screen of the BMS 3270 transaction as it would appear on a BMS 3270 terminal, with additional graphical elements overlaid onto the contents of the screen.

The contents of the screen are interactive, allowing you to highlight or otherwise select fields and make changes such as naming fields and setting default values and field parameters.

Input fields recognized by SOLA are shown as solid green boxes.

Whenever you move the mouse cursor over a field (input, output, etc.), the cursor changes from an arrow to a hand (this may vary if you have custom cursor settings).

Field Settings



When the cursor changes to a hand over a field, you can right-click to access a menu of options for that field. Each field has its own menu of options that define it and its role in the transaction.

The menu contains the following options (some options are not displayed with some fields):

Input Value: this is the input value you want to enter to run the transaction. Even though in the execution of the web service this value may be dynamic, SOLA has to be taught to run the transaction with fixed values. The generated WSDL can then be used to run the transaction with input from user or application. The value is going to be same as what you would enter on a legacy green screen for this field in order to execute the transaction. This option is only present if SOLA determines that the associated field is an input field.

Field Name: this is used to assign a name to the field that a requestor of this service would see. This is the name that will be published in the WSDL.

> **NOTE:** If you change any properties of a field, SOLA will not allow you to proceed unless you change the field name. The word "Field" with an uppercase F cannot be a part of that name, as "Field' is a restricted word. The name cannot contain spaces.

Data Type: this is used to specify the data type of an input or output field. Options are string, int, short and decimal. This applies only to the WSDL file generated by SOLA and not to the operations on the mainframe side. For example, if a certain field always displays a number and the mainframe code identifies it as a string, you can set it to integer and the consumer of the web service will treat the field as an integer.

IO: this is used to specify the nature of a field. It can have one of the following values:

Input: indicates that requestor will send this field to SOLA. SOLA may then use the field to populate green screen, if the screen allows it.

Output: indicates the SOLA will pick this value from green screen and send it to requestor.

InputOutput: indicates field is Input as well as Output.

Exclude: indicates that a request will not send data related to this field. However, SOLA may decide to put a hidden value if MDT is set to ON.

ErrorMsg: used in conjunction with the Repeat setting in the BMS Analyzer tools (left side of screen). If the map is defined as repeating two times but during execution SOLA encounters the same screen three times, SOLA would send a SOAP Fault (error message). This error message will be picked from the field that is defined as "ErrorMsg".

AlwaysDefault: indicates that SOLA will not publish this field to the requestor, and will instead use a default value entered during this analysis as input during real execution.

EndMarker: is used in conjunction with the "Unlimited" Repeat setting in the BMS Analyzer tools (left side of screen) in order to allow SOLA to stop the execution at the desired point. With the MAP set to "Repeat Unlimited", SOLA needs a way to stop the transaction. Setting an EndMarker value indicates to SOLA it should end the transaction if the specified value (the value of the field when an EndMarker was set) is encountered during real execution.

ContinueMarker: is the opposite of EndMarker and is used in conjunction with the "Unlimited" Repeat setting in the BMS Analyzer tools (left side of screen) in order to allow SOLA to stop the execution at the desired point. With the MAP set to "Repeat Unlimited", SOLA needs a way to stop the transaction. Setting an ContinueMarker value indicates to SOLA it should end the transaction if the specified value is not the value of the field during real execution.

DrillDown: this is an input field that can be used to drill down (retrieve information about) another field.

Key: this is a field that is used to identify a specific data item. In a list of data items, the key field will be the field used to match a search query with the data item being searched for. For example, when searching a list of widgets for a specific widget, you can use the widget number as the key field. The user would pass a widget number, and the transaction would scroll the list of widgets until a widget with matching widget number is found.

MDT: indicates if MDT on the screen is ON or OFF. This value cannot be changed by the user.

Protected: indicates if the field is protected. Possible values are YES and NO. This value cannot be changed by the user.

Hidden: indicates if the field is hidden. Possible values are YES and NO. This value cannot be changed by the user.

Len: indicates the character length of the field. This value cannot be changed by the user.

Instance: this value is a counter indicating the repeat instance of the map (how many times it has been repeated during the transaction). For example, a blank screen counts as one

instance, and each time data is entered and the Enter key is pressed counts as another instance.

StartOfGroup: SOLA populates this value only for the first field out of multiple fields selected and "Grouped as Output" fields (see Marquee Tool section).

GroupParent: this is used only with fields that have been grouped (groups are explained later in this section). This value will become a parent element in the output XML. All the fields that are grouped together would appear as children elements under this name in the output XML.

For example, in the following image there are 7 fields:

All of these fields are grouped together and have a single parent called "ItemDetail".

In this case output XML would appear as follows:

```
<ItemDetail>
    <child1>X</child1>
    <child2/>
    <child3>09/06/05</child3>
    <child4>WCMTD</child4>
    <child5/>
    <child6>100.04</child6>
    <child7>WCM # 0165044</child7>
</ItemDetail>
```

To close the menu without saving changes, click CANCEL .

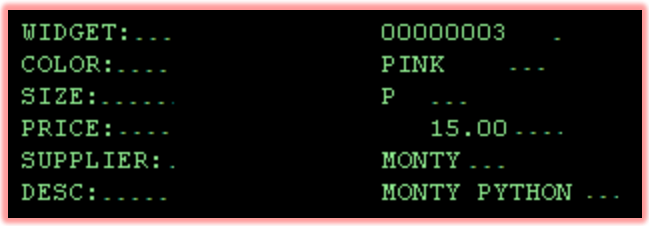
To hide the menu and save changes, click APPLY .

For information about the TRANSLATE button, see "Translation Feature" below.

Translation Feature

For dealing with cryptic or inadequate data labels, SOLA offers the translation feature.

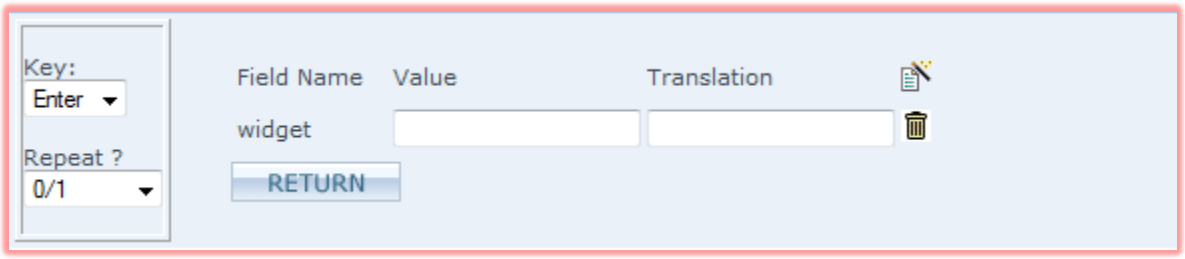
Consider the following display:



For size, the data indicates a value of "P", which means petite. However, since the letter P is not usually associated with size, you may choose to pass a value to the data recipient that is more easily understand. To accomplish this, SOLA offers the translation feature.

To access the translation feature, right click on the desired field to display the field options menu, then click the TRANSLATE button.

Doing so will display the translation sub-panel.

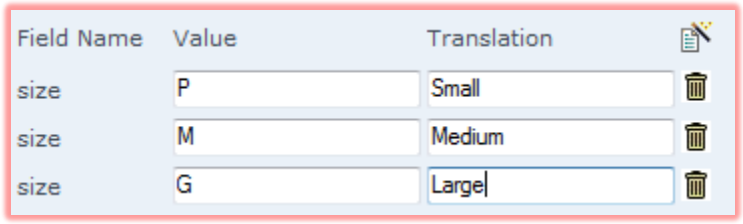


The sub-panel consists of three columns, one of which is pre-populated with the parent field name. To use the translation feature, enter a value (from the program) under the Value column, then enter a translation for the value in the Translation column. To add more blank fields (to enter more values), click the button. To remove unnecessary blank fields, click the button.

When the web service is executed, the translated value will be sent to the user instead of the original value.

For example, the possible values for size are P, M and G (Petite, Medium and Grand). Since these are obscure and unintuitive, we want to replace them with the more easily understood values of Small, Medium and Large. To translate these values, right-click on the Size field to display the field options menu and then the TRANSLATE button.

Since there are three values, click the button twice to add two more fields. Enter the three values in the blank fields under the Value column, then their translations under the Translation column.

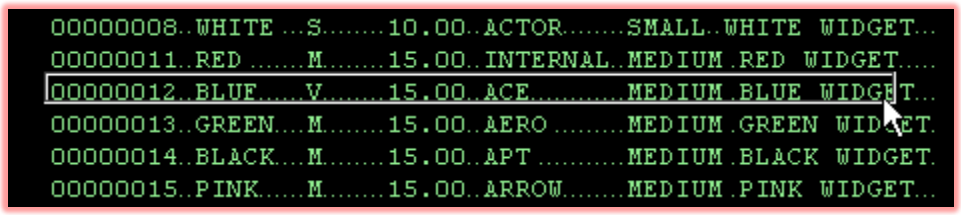


When finished, click RETURN. Whenever the web service is executed, the value P will be replaced with Small, M with Medium and G with Large.

Marquee Tool

The marquee (or selection) tool is one of the most powerful features of SOLA's BMS 3270 Analyzer. Using the tool is similar to using a marquee tool in a graphics program that is used to select a portion of an image.

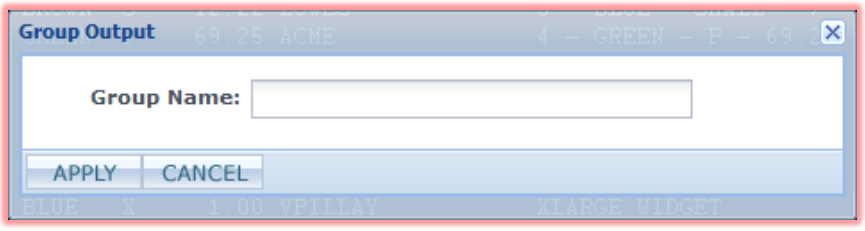
To use the marquee tool, double click in the top left corner of what you want to select, then double click the bottom right corner. A rectangle will appear, with its top left corner located at the exact point of your first double click, and its bottom right corner in the exact location of your second double click.



Selected items will appear with a white background. It is important to note that the marquee does not represent the selected fields, the white background does. The marquee is a guide for selecting the fields, nothing more.

Once a selection is made, a menu of options appears:

Group Output: combines the selected fields into a



group. In the output xml, the fields will be listed under a group heading and can be manipulated as a single group by the output data consumer. The first field in a group will be designated as the group parent. When this option is selected, the following dialog box appears, allowing you to name the group and specify it's type.

Merge Output: combines the selected output fields into a single string. This is useful when you want to send a group of fields as a single string. SOLA will accept the discrete fields from the program and combine them into a single string during execution.

Merge Input: combines the selected input fields into a single string. This is useful when you may have multiple input fields on the green screen but you want the requestor to send a single string rather than multiple strings. SOLA will chop the input strings and fill the necessary multiple fields during execution.

Copy: this is a very powerful feature of the marquee tool. It allows you to make changes to a field, then copy those changes and paste them to other fields. Using this feature, you only have to make one set of changes when working with a large list of repetitive fields. To use this feature, make changes to a field, select it with the marquee tool, then select Copy Modifications. Use the marquee too to select a single field or a group of fields that you want to share the same settings, then select Paste.

Paste: select this option to paste the settings of one or more fields copied using the Copy Modifications option onto a matching field or group of fields. The target selection must match the copied fields. For example, if a line has four fields and you copy the first three, you can only paste them onto the first three fields of every remaining line.

Split Field: this is another very powerful feature of the marquee tool that lets you split a single large field into several smaller fields. This feature is explained later in this section.

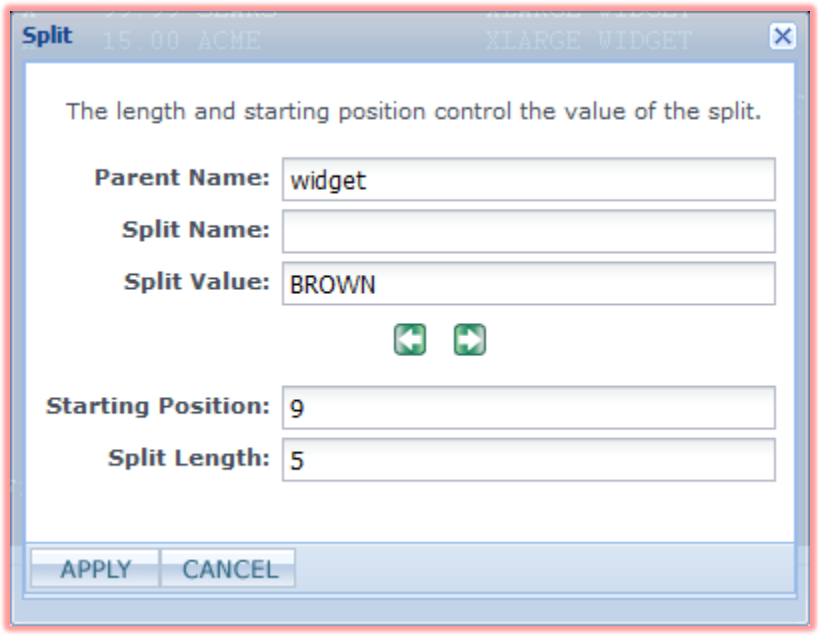
Cancel: cancels the selection.

Split Field Feature

The Split Fields feature allows a single long string containing several meaningful pieces of data to be sent as separate data items. This relieves the consumer of the burden of having to worry about formatting/reformatting issues.

To use the feature, first use the Field Options menu to name and configure the field you want to split. Then select a portion of the field with the marquee tool and choose Split Field from the marquee tool options menu.

This will display the Split Field menu.



Parent Name: this value is populated by SOLA and is taken from the name given to the parent field using the Field Options menu. This value cannot be changed by the user (except in the Field Options menu).

Split Name: use this to assign a name to the split portion of the field.

Split Value: this is the field value, which SOLA attempts to retrieve from your marquee tool selection. Although you can manually alter this value, it is recommended that you use the Starting Position and Split Length fields to adjust it.

Starting Position: allows you to set an offset value (number of spaces) for the split field. This is used in conjunction with Length to pinpoint the correct location of the split field. You know you have the correct offset and length settings when the Split Value field displays the correct value (of the portion of the field you are trying to split).

Split Length: allows you to set a length value (in characters) for the split field. This is used in conjunction with Offset to pinpoint the correct location of the split field. You know you have the correct offset and length settings when the Split Value field displays the correct value (of the portion of the field you are trying to split).

To save changes and split the field, click APPLY.

To exit without saving changes, click CANCEL.

Repeat the process for all portions of the field you want to split.

Key Matching Feature

A key field is a field that is used to identify a specific data item. In a list of data items, the key field (or fields) will be the field(s) used to match a search query with the data item being searched for. For example, when searching a list of widgets for a specific widget, you can use the widget number as the key field. The user would pass a widget number, and the transaction would scroll the list of widgets until a widget with matching widget number is found.

There are two ways to use key matching; drill down and update.

Drill Down Key Matching

To use the key field for drill down key matching, you must set the following values:

Key: this key advances screens when NOT scrolling through a list trying to match the key field. This can be the same as the Scrolling Key, but doesn't have to be.

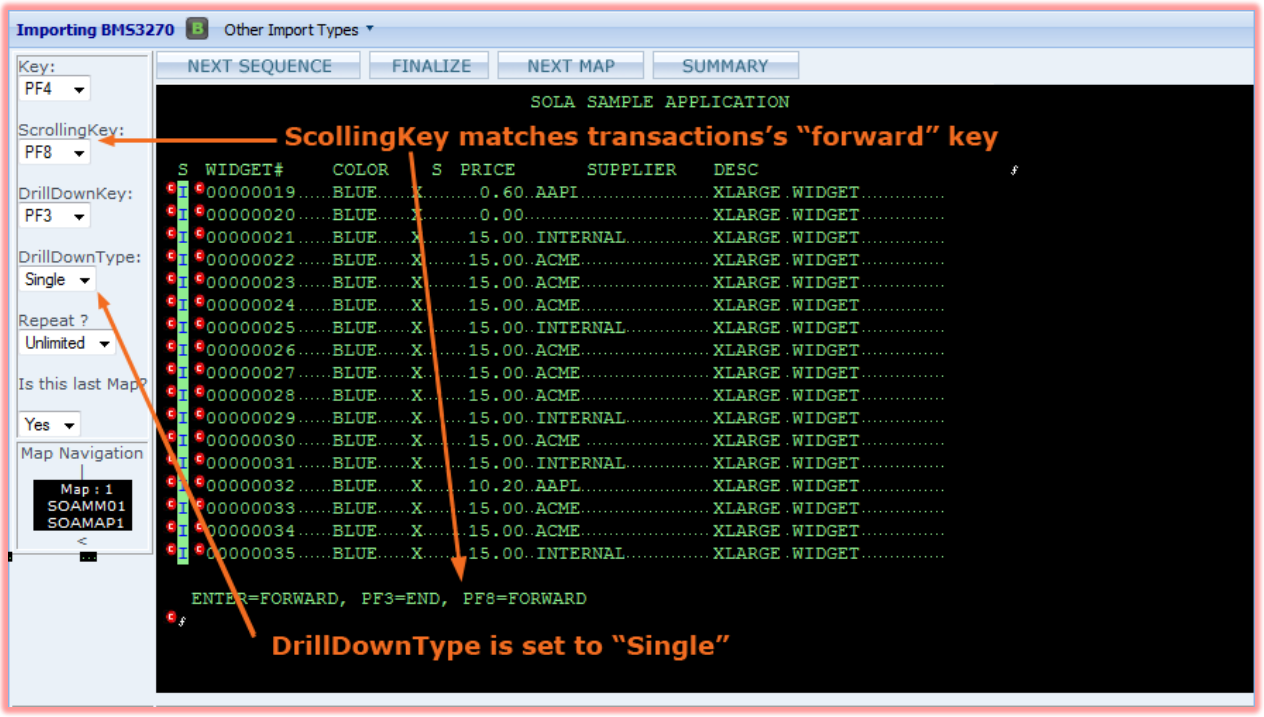
Scrolling Key: this must be set to the key the transaction needs to scroll through a list while trying to match the key field.

Drill Down Key: this must be set to the key the transaction requires to drill down when a matching field has been found (key field match successful).

Drill Down Type: this must be set to single, so that SOLA knows that only list items that match the key field should be drilled down.

Repeat: this should be set to "Unlimited" so that the transaction can scroll down as far as necessary to match the key field.

Is this the Last Map: this should be set to "Yes" so that SOLA does not enter an endless loop if the map doesn't match.



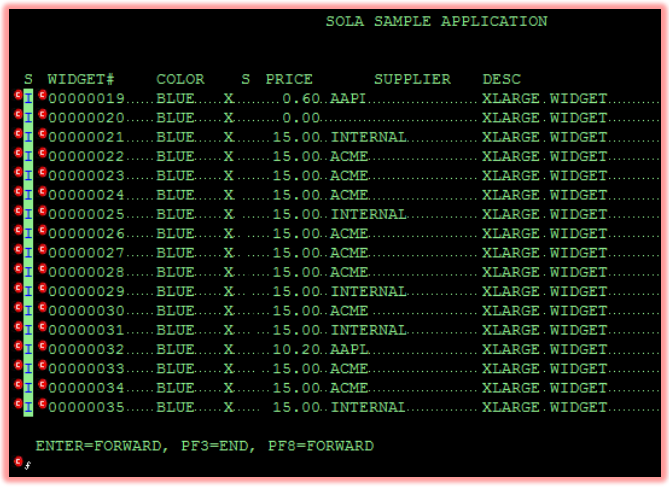
You must also define the following fields on the map:

Drill Down Field: this is the field, typically located at the head of a row, in which the transaction requires a certain input (e.g. "i") to drill down into its associated row.

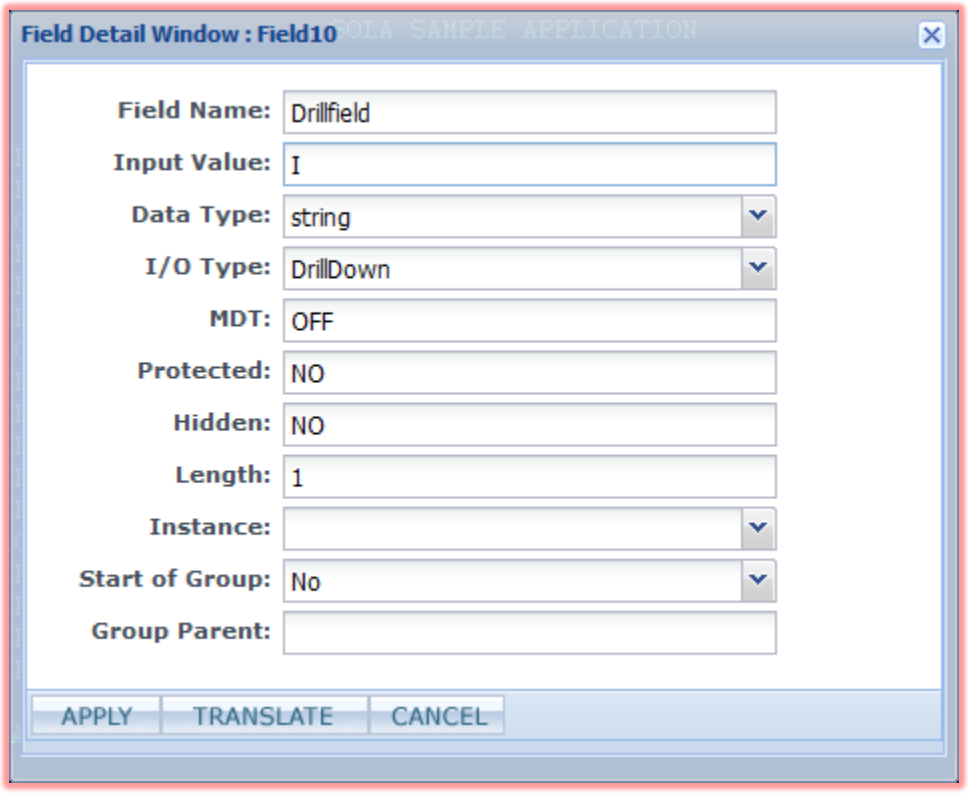
Key Field: this is the field or fields in the row that will be matched to the search query.

Continue Marker or End Marker: define a continue marker as a precaution to allow SOLA to stop scrolling if a match is not found.

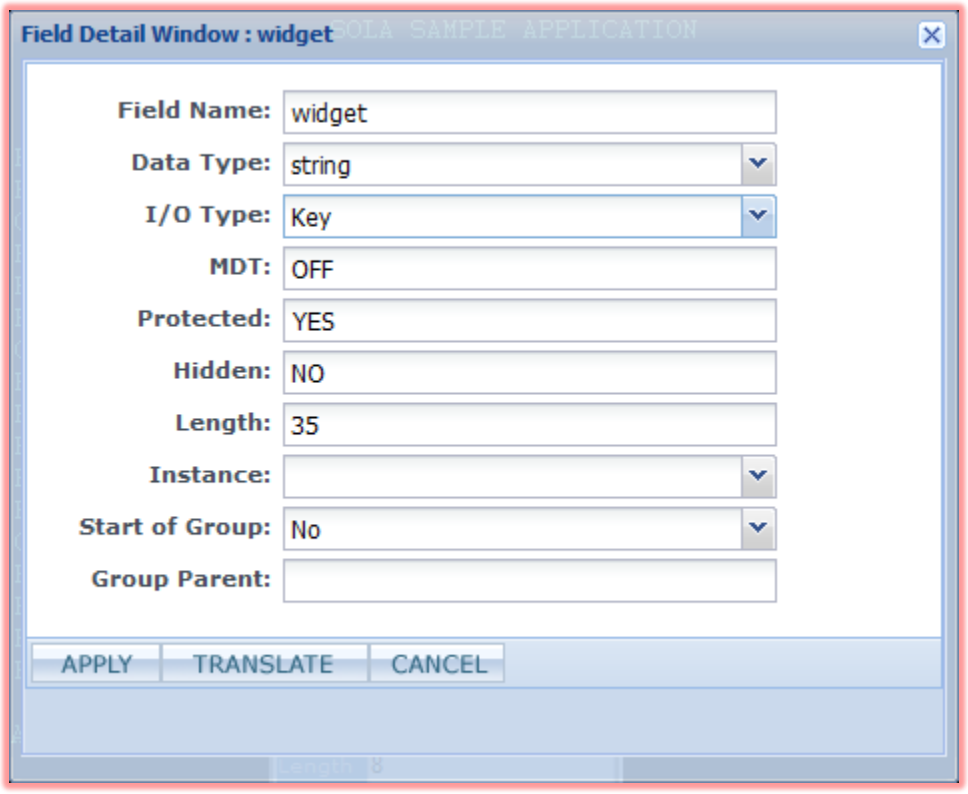
For example, the following screen contains a list of widgets:



The leading input field has been defined as a Drill Down field:



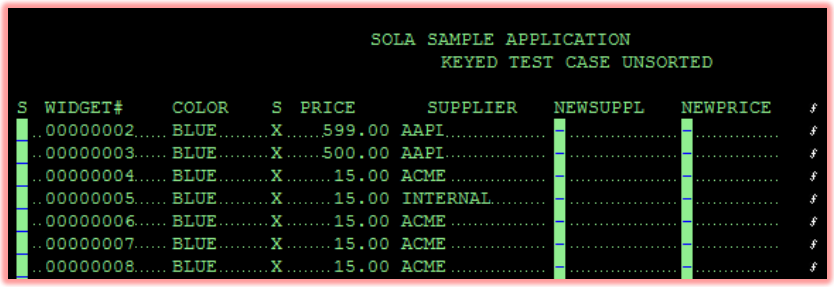
The Widget number field has been defined as the Key Field:



Once the drill down field and the key field have been defined, advance to the next map (the results of the drill down). Once that map is configured, return to the list screen and copy and paste the settings for the drill down and key fields all the way down the list. Since **DrillDownType** was set to single, SOLA will only drill down if the key field matches.

Update Key Matching

Key matching can also be used to update data (rather than just drill down for more information). For example, a widget transaction might have one or more input fields on every line of data:



In this transaction, you search for a matching widget number then enter a new supplier and a new price. The settings for this type of transaction are the same as the previous use of key matching, except that you do not define a drill down field on the map. Instead, you enter values in the input fields then set the **DrillDownKey** and **Key** values to the keyboard key that the transaction wants as an update key and click **Next Map**.

Once you are at the next screen (typically it's the first screen) with an "Update Successful" message, go back to the list screen and make the **Key** value different from the **DrillDownKey** value.

Caution: before finalizing, verify the **Key**, **ScrollKey**, **DrillDownKey** and **DrillDownType** values.

Graphics View Screen Symbols

The following symbols may appear next to fields on the screen.

This symbol indicates that changes were made to the field (using the field settings menu).

This symbol indicates that the field has been split into component features with the Split Field feature (available through the Marquee tool).

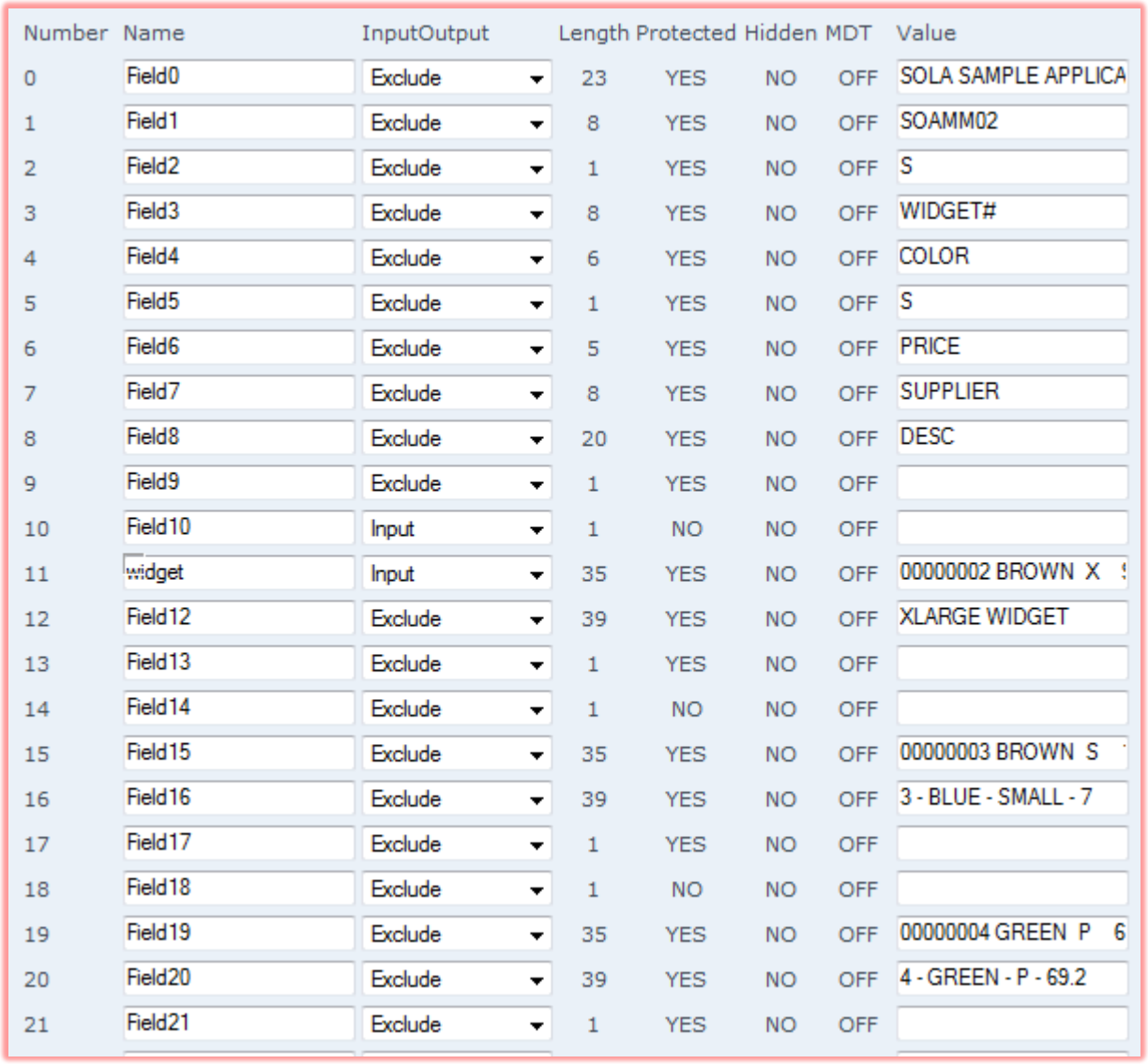
This symbol indicates a part of a field that has been split into a separate component field with the Split Field feature (available through the Marquee tool). This symbol appears in conjunction with the scissors symbol, and only in rows where you have manually split the row. If you copy and paste field settings it will only show up in original (see the Marquee Tool section for information on copying and pasting).

This is not a discrete symbol but is in fact an overlap of the scissors and arrow symbols that sometimes occurs on the screen.

This symbol indicates the presence of a field whose value has not been set. This can be used to spot unpopulated or empty fields.

Working with the Fields View

The 'Fields View' displays the BMS 3270 screen as a series of fields with associated options. This view is not as versatile or powerful in terms of features as the graphics view, but can be used to quickly make changes to all of the fields of the screen in one place.



Number	Name	InputOutput	Length	Protected	Hidden	MDT	Value
0	Field0	Exclude	23	YES	NO	OFF	SOLA SAMPLE APPLICA
1	Field1	Exclude	8	YES	NO	OFF	SOAMM02
2	Field2	Exclude	1	YES	NO	OFF	S
3	Field3	Exclude	8	YES	NO	OFF	WIDGET#
4	Field4	Exclude	6	YES	NO	OFF	COLOR
5	Field5	Exclude	1	YES	NO	OFF	S
6	Field6	Exclude	5	YES	NO	OFF	PRICE
7	Field7	Exclude	8	YES	NO	OFF	SUPPLIER
8	Field8	Exclude	20	YES	NO	OFF	DESC
9	Field9	Exclude	1	YES	NO	OFF	
10	Field10	Input	1	NO	NO	OFF	
11	widget	Input	35	YES	NO	OFF	00000002 BROWN X
12	Field12	Exclude	39	YES	NO	OFF	XLARGE WIDGET
13	Field13	Exclude	1	YES	NO	OFF	
14	Field14	Exclude	1	NO	NO	OFF	
15	Field15	Exclude	35	YES	NO	OFF	00000003 BROWN S
16	Field16	Exclude	39	YES	NO	OFF	3 - BLUE - SMALL - 7
17	Field17	Exclude	1	YES	NO	OFF	
18	Field18	Exclude	1	NO	NO	OFF	
19	Field19	Exclude	35	YES	NO	OFF	00000004 GREEN P 6
20	Field20	Exclude	39	YES	NO	OFF	4 - GREEN - P - 69.2
21	Field21	Exclude	1	YES	NO	OFF	

Field Num: a sequential value assigned to all fields on a given map, starting with the first field detected.

Field Name: this is used to assign a name to the field that a requestor of this service would see. This is the name that will be published in the WSDL.

InputOutput: this is used to specify the nature of a field and can have one of the following values:

Input: indicates that requestor will send this field to SOLA. SOLA may then use the field to populate green screen, if the screen allows it.

Output: indicates the SOLA will pick this value from green screen and send it to requestor.

InputOutput: indicates field is Input as well as Output.

Exclude: indicates that a request will not send data related to this field. However, SOLA may decide to put a hidden value if MDT is set to ON.

ErrorMsg: used in conjunction with the Repeat setting in the BMS Analyzer tools (left side of screen). If the map is defined as repeating two times but during execution SOLA encounters the same screen three times, SOLA would send a SOAP Fault (error message). This error message will be picked from the field that is defined as "ErrorMsg".

AlwaysDefault: indicates that SOLA will not publish this field to the requestor, and will instead use a default value entered during this analysis as input during real execution.

EndMarker: is used in conjunction with the "Unlimited" Repeat setting in the BMS Analyzer tools (left side of screen) in order to allow SOLA to stop the execution at the desired point. With the MAP set to "Repeat Unlimited", SOLA needs a way to stop the transaction. Setting an EndMarker value indicates to SOLA it should end the transaction if the specified value is encountered during real execution.

ContinueMarker: is the opposite of EndMarker and is used in conjunction with the "Unlimited" Repeat setting in the BMS Analyzer tools (left side of screen) in order to allow SOLA to stop the execution at the desired point. With the MAP set to "Repeat Unlimited", SOLA needs a way to stop the transaction. Setting an ContinueMarker value indicates to SOLA it should end the transaction if the specified value is not the value of the field during real execution.

Length: indicates the character length of the field. This value cannot be changed by the user.

Protected: indicates if the field is protected. Possible values are YES and NO. This value cannot be changed by the user.

Hidden: indicates if the field is hidden . Possible values are YES and NO. This value cannot be changed by the user.

MDT: indicates if MDT on the screen is ON or OFF. This value cannot be changed by the user.

Value: this is the value of the field taken from the green screen. Making changes to the value usually doesn't affect anything. The only reason for you to change the value is for use with the following settings; EndMarker, ContinueMarker or AlwaysDefault.

Field View Screen Symbols

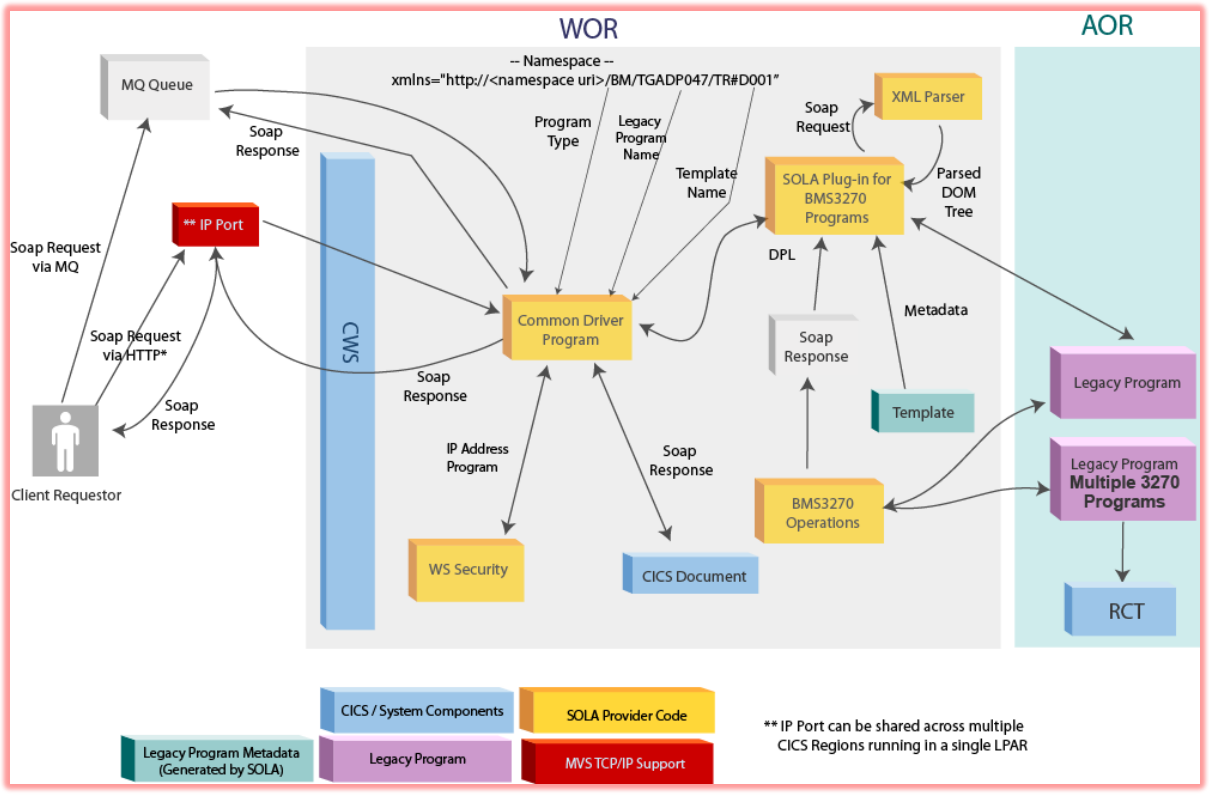
The following symbols may appear next to fields on the screen.

This symbol indicates that changes were made to the field (using the field settings menu).

This symbol indicates that the field has been split into component features with the Split Field feature (available through the Marquee tool).

Environment Setup

Before you can test a new web service you will need to perform some set-up steps for the SOLA Run-time. On page 47, we discussed the MRO (Multiple Region Operation) concept. The following is an MRO diagram of the architecture for 3270 programs.



SOA Enabling BMS3270 via CWS/MQ

To configure the WOR and AOR so that our Widget transaction will run, we will need to create the following definitions.

WOR	AOR
DEFINE PROGRAM(XMLPCWGT) GROUP(SOLAGRP) LANG(LE)	EXISTING TRANSACTION (TGW#)
DEFINE PROGRAM(TGW#D001) GROUP(SOLAGRP) LANG(ASSEMBLER) STATUS(ENABLED)	
DEFINE TRANSACTION (TGW#) REMOTESYSTEM(AOR)* REMOTENAME(TGW#)	

For this example, the Transaction will be running in the WOR and so will not need a Remotesystem or a Remotename.

We define a "dummy" program that will drive the transaction in the WOR. We also define the template (an assembler program) in the WOR. Finally, a transaction is set up in the WOR pointing to the AOR in which the BMS 3270 transaction runs.

Using SOLA Developer – Stored Procedures

Stored procedures, sometimes called sprocs or SPs, are subroutines stored in databases that can be called by applications. They are most often used for data validation, access control and to consolidate functions that were originally implemented in applications.

How SOLA Creates Web Services from Stored Procedures

SOLA is capable of registering stored procedures and making them available as web services. The registration process involves searching for stored procedures, supplying necessary arguments, executing the stored procedure and finalizing registration. Once registered, a stored procedure becomes a method stored in SOLA's UDDI directory and can be called as a web service.

Creating a Web Service from a Stored Procedure

This section will describe the steps necessary to create a web service from a stored procedure by searching for a specific stored procedure, registering it with the SOLA directory, providing required arguments, executing the stored procedure and finalizing the registration. There is no analysis when making web services from stored procedures, as procedures are fairly simple and perform single functions. The end result will be a WSDL, metadata template, test harness and a UDDI entry.

Step 1 – Mainframe Preparation

Before you begin the stored procedure registration process, it is a good idea to configure the mainframe environment to enable your stored procedure to be executed. During the registration process, SOLA will attempt to execute the stored procedure to verify that the data you have provided is valid. Although you can set up the environment at any time before the actual execution takes place, it is a good idea to do so before the registration process.

To configure the mainframe environment, you will need to set up PPT entries in the SOLA WOR.

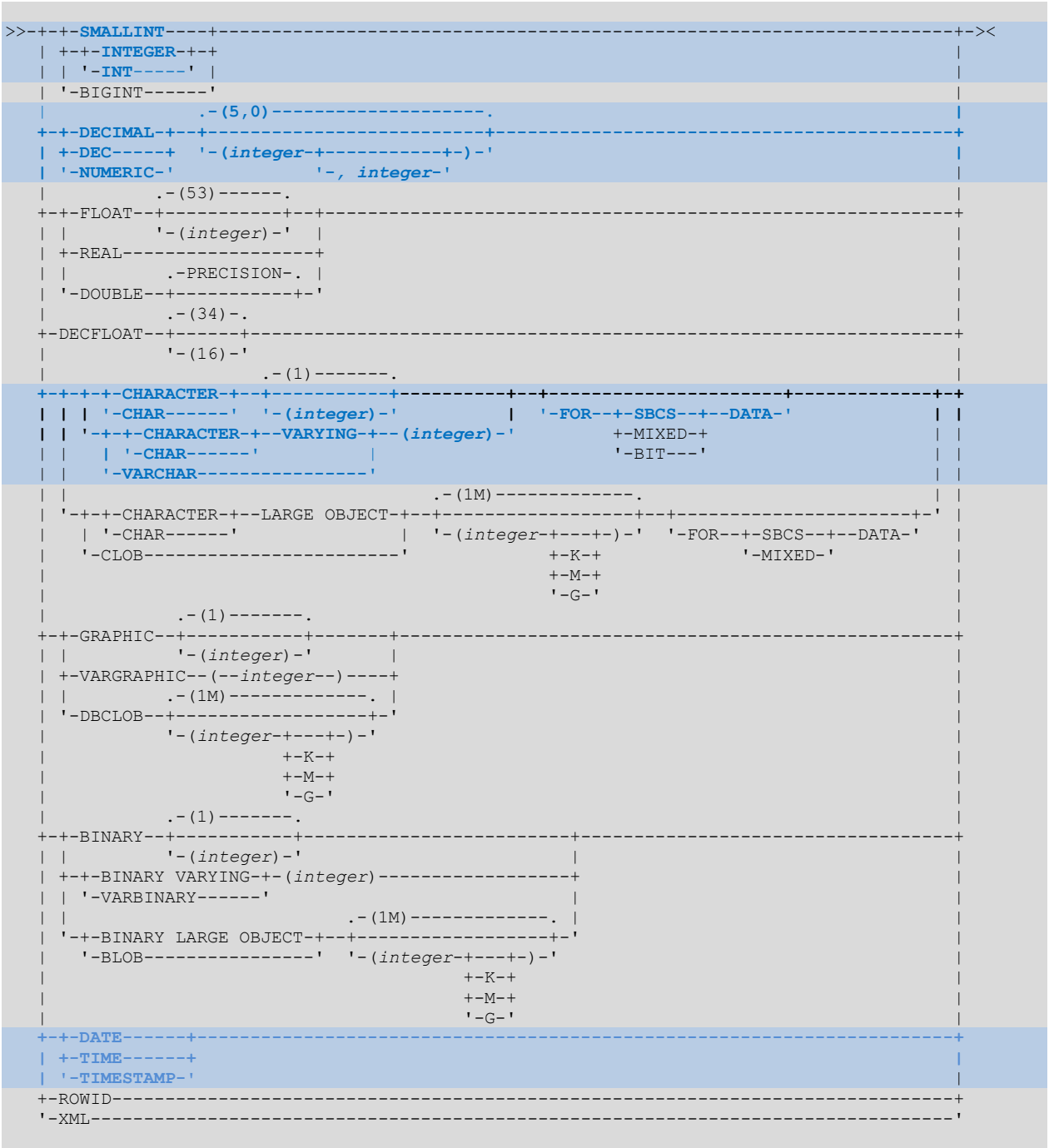
WOR	**AOR**
PPT: DEFINE PROGRAM(yourSPprogName) LANG (LE) STATUS (Enabled) REMOTE REGION(yourRegion) REMOTE NAME:XMLPC200 REMOTE TRANID: (yourTranId)	**PPT:** DEFINE PROGRAM(XMLPC200) LANG (LE) STATUS (Enabled)
	PCT: DEFINE TRANSACTION(yourTranId) PROGRAM (DFHMIRS) PROFILE (DFHCICSA) STATUS (Enabled)
	RCT: DB2ENTRY(yourTranId) PLAN (XMLPLAN)*

* XMLPLAN must contain an entry which represents the collection that your target stored procedure belongs to.

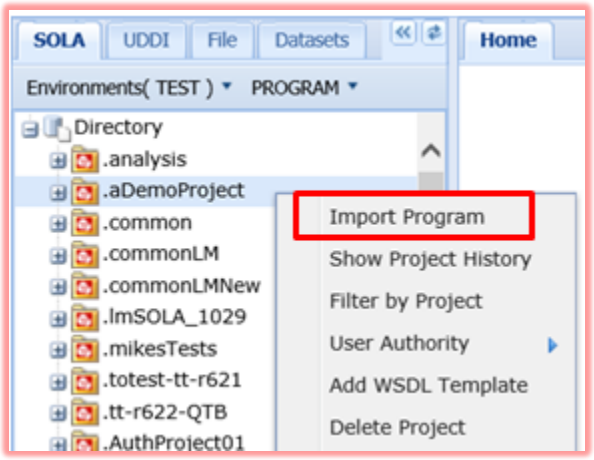
If you have any difficulties with making these table entries, consult an administrator.

Step 2 – Verify Stored Procedure Syntax

The SOLA run time supports a subset of the stored procedure syntax, shown below in **blue**. Before importing a stored procedure, verify that your procedure expects or returns only the supported data types:



Step 3 – Stored Procedure Registration

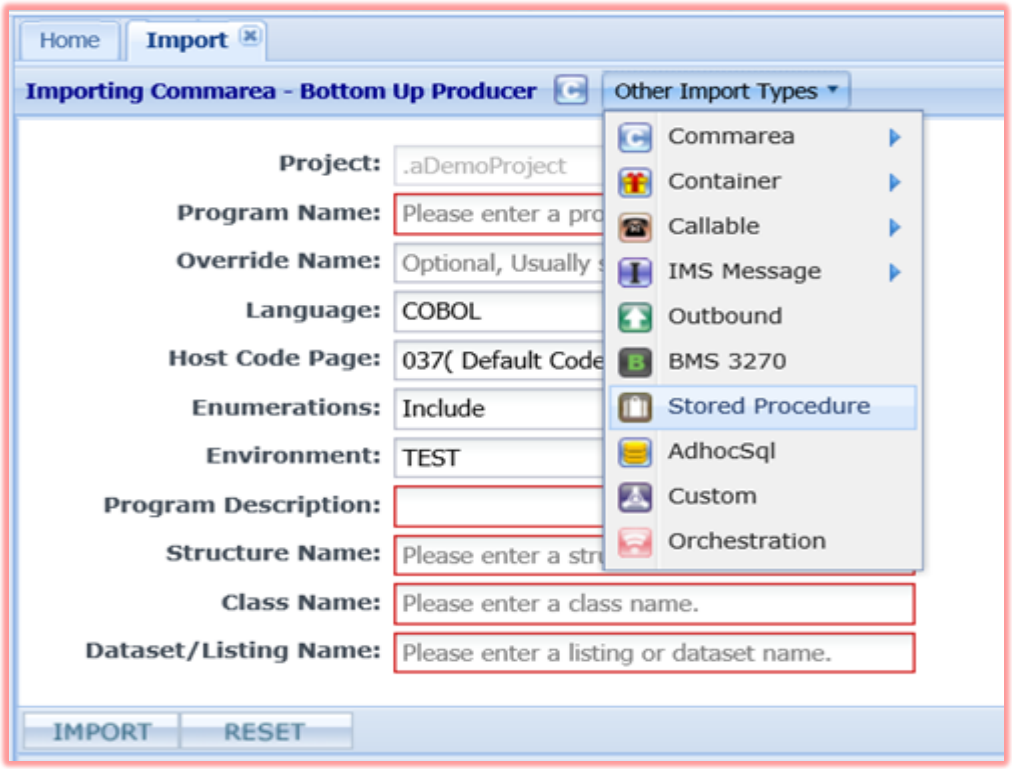


Select the project you wish to import to and right-click it. From the pop-up menu, select **Import Program**. If you wish to import the program to a new project, first follow the steps for creating a new project on page 22.

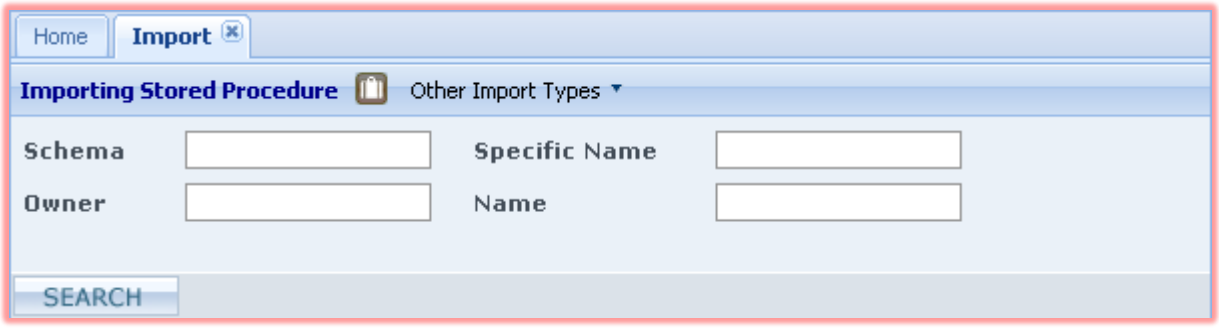
In order to import a program into a project, you must be an authorized user of that project. Once the program is imported, you can drag and drop the program from one project to another. However, you must also be an authorized user of the project you wish to move the program into.

After you select **Import Program**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

The default program type is commarea, so use the **Other Import Types** menu to select Stored Procedure.



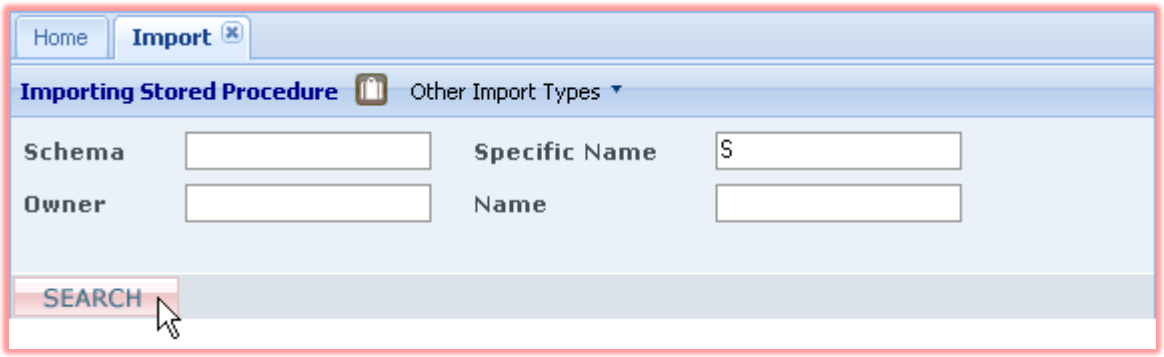
The Import panel will change to display the stored procedure search panel.



The first step in creating a web service from a stored procedure is to select which stored procedure you want to use. To do so, you search for it based on one of the following search criteria:

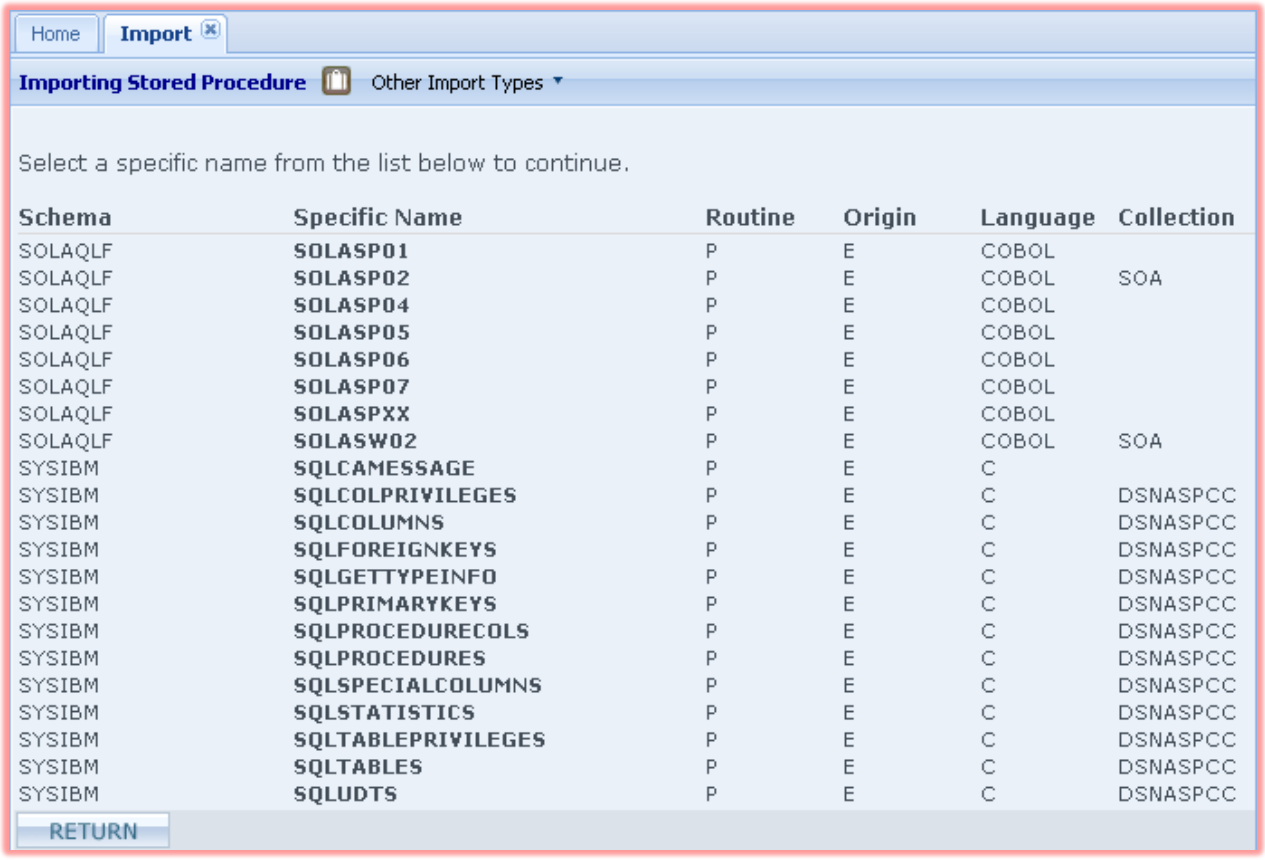
- **Schema:** narrows the search to stored procedures with a matching schema.
- **Specific Name:** narrows the search to stored procedures with a matching name.
- **Owner:** narrows the search to stored procedures with a matching owner name.
- **Name:** narrows the search to stored procedures with a matching alias name (usually the same as specific name).

Although all fields are optional, you must supply at least one parameter. Wildcard characters (%) are permitted during the search, as are partial words. The example blow search below is based around a single letter of the stored procedure's specific name, so all stored procedures whose specific name starts with the letter "s" will be returned. If you wanted to return all results that have the letter s in the specific name (rather than just those that start with s), you could specify %s in the **Specific Name** field.



Once you have specified at least one search parameter, click 

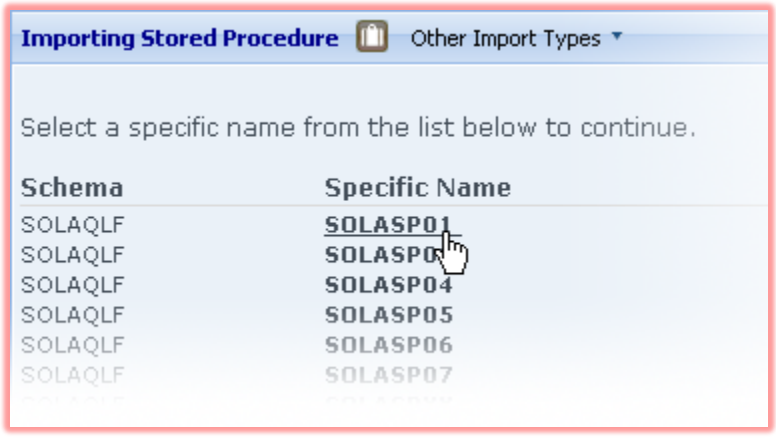
. The results summary panel will be displayed.



Schema	Specific Name	Routine	Origin	Language	Collection
SOLAQLF	SOLASP01	P	E	COBOL	
SOLAQLF	SOLASP02	P	E	COBOL	SOA
SOLAQLF	SOLASP04	P	E	COBOL	
SOLAQLF	SOLASP05	P	E	COBOL	
SOLAQLF	SOLASP06	P	E	COBOL	
SOLAQLF	SOLASP07	P	E	COBOL	
SOLAQLF	SOLASPXX	P	E	COBOL	
SOLAQLF	SOLASW02	P	E	COBOL	SOA
SYSIBM	SQLCAMESSAGE	P	E	C	
SYSIBM	SQLCOLPRIVILEGES	P	E	C	DSNASPCC
SYSIBM	SQLCOLUMNS	P	E	C	DSNASPCC
SYSIBM	SQLFOREIGNKEYS	P	E	C	DSNASPCC
SYSIBM	SQLGETTYPEINFO	P	E	C	DSNASPCC
SYSIBM	SQLPRIMARYKEYS	P	E	C	DSNASPCC
SYSIBM	SQLPROCEDURECOLS	P	E	C	DSNASPCC
SYSIBM	SQLPROCEDURES	P	E	C	DSNASPCC
SYSIBM	SQLSPECIALCOLUMNS	P	E	C	DSNASPCC
SYSIBM	SQLSTATISTICS	P	E	C	DSNASPCC
SYSIBM	SQLTABLEPRIVILEGES	P	E	C	DSNASPCC
SYSIBM	SQLTABLES	P	E	C	DSNASPCC
SYSIBM	SQLUDTS	P	E	C	DSNASPCC

The results summary shows information about all of the stored procedures that match your search criteria. The information is organized under the following column headings:

- **Schema:** the stored procedure's schema name.
- **Specific Name:** the stored procedure's internal name. The names in the column are links. Click on the link to select that stored procedure and create a web service that will call it.
- **Routine:** the routine type, can be P (for procedure) or F (for function).
- **Origin:** the stored procedure's origin, can be E (for external) or I (for internal).
- **Language:** the stored procedure's language. Options are all types, Assemble, C, Cobol, Compjava, Java, PLI, Rexx and SQL.
- **Collection:** the DB2 collection that the stored procedure belongs to.

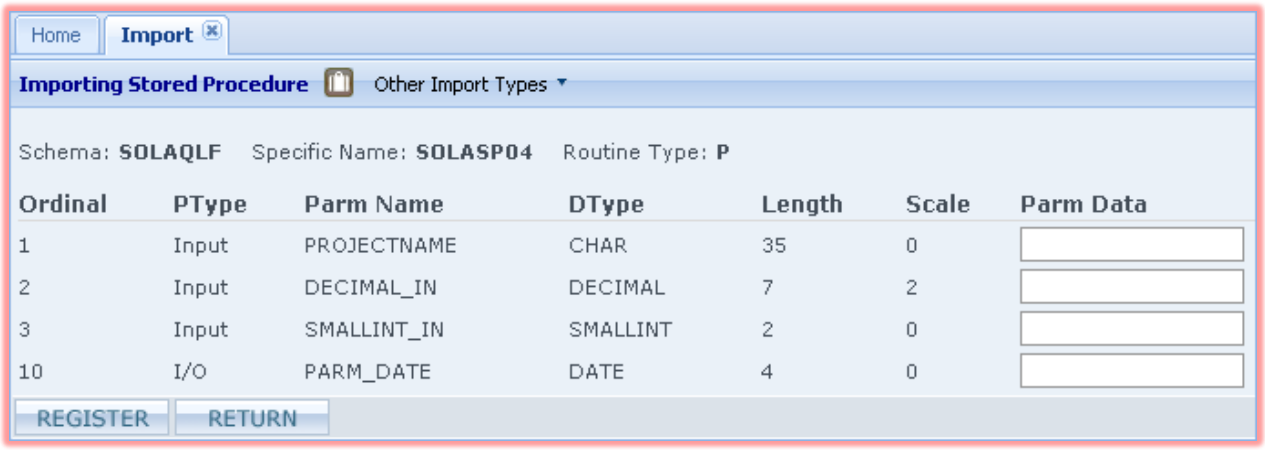


To continue with the import process, select a stored procedure from the list by clicking on its name in the **Specific Name** column.

If no stored procedures are listed, go back to the previous screen and try a new search with different parameters.

At any point during import, you can return to the previous panel by clicking RETURN.

Clicking on a stored procedure's name will display the procedure details panel.



The procedure details panel displays the input portion of the signature for the selected stored procedure. In order to execute, most stored procedures require input. SOLA will analyze the selected stored procedure and automatically determine what input fields it requires. You will need to input the data required to execute the stored procedure before proceeding.

The data you input will not necessarily be the data that is used when the stored procedure is called as a web service after registration. SOLA needs valid input data to execute the stored procedure to make sure that it was registered correctly.

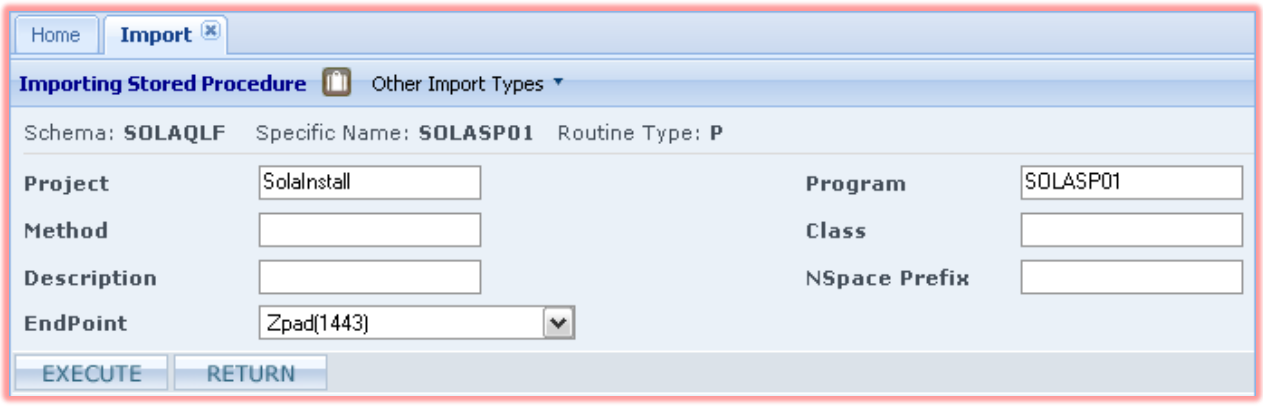
The panel provides information about each parameter in the input portion of the stored procedure's signature under a series of column headings:

- **Ordinal:** the order that the parameter appears in the signature.
- **PType:** the parameter type, either I (input) or I/O (input/output).
- **Parm Name:** the parameter name.
- **DType:** the type of data that this parameter represents.

- **Length:** the parameter's internal length.
- **Scale:** for some types of parameters (such as fractions), the scale represents the number of significant positions after the decimal point.
- **Parm Data:** this column contains fields in which parameter values can be entered.

To proceed, enter the appropriate values (you will need to be familiar with the stored procedure and what it does to know what values are appropriate) for every parameter and click REGISTER to continue.

The registration panel will be displayed.



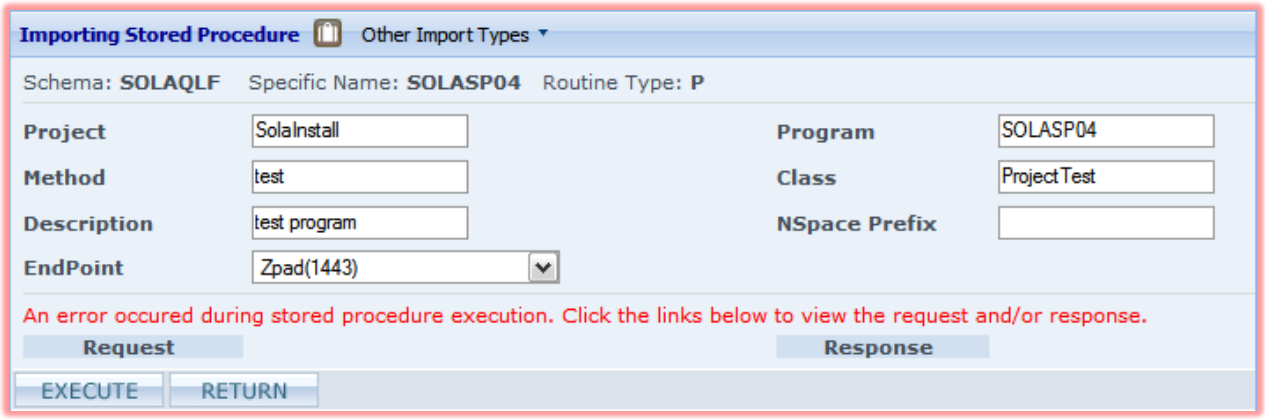
The registration panel is where you give SOLA the information it needs to expose the stored procedure as a web service. The stored procedure must be given a method, class and program name. You can also specify the end point, an optional namespace prefix and provide a description.

- **Project:** the SOLA project that the stored procedure will belong to. This cannot be changed during registration, though you can drag the program from one project to another once it is created.
- **Program:** the SOLA program name that the stored procedure will be organized under. This cannot be changed.
- **Method:** the SOLA method name used to execute the stored procedure.
- **Class:** the SOLA class name that the stored procedure will be organized under.
- **Description:** a free-form description field.
- **NSpace Prefix:** This field is optional. Use it to customize the descriptive portion of the namespace of the WSDL that will be generated when the stored procedure is registered.
- **EndPoint:** the end point that the stored procedure will run in.

When you have provided the registration information, click EXECUTE .

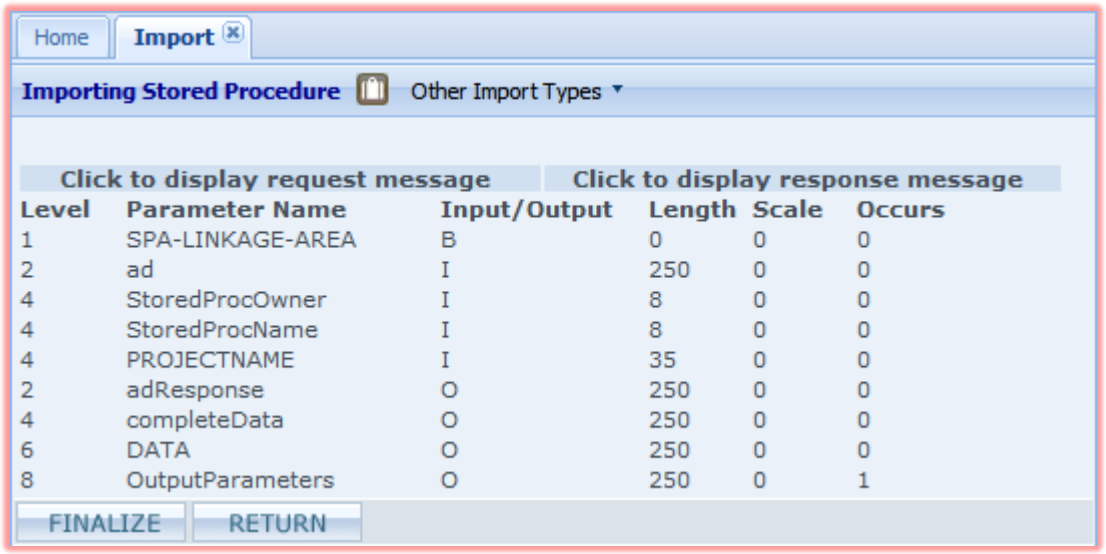
SOLA will attempt to execute the stored procedure with the data you provided (on the Stored Procedure Details screen) and display the results.

If the stored procedure does not execute successfully, an error message will indicate failure, and SOLA will display a new set of hyperlinks at the bottom of the panel.



Clicking on the Request link will open a new window containing the SOAP request that was sent when SOLA attempted to execute the stored procedure. Clicking the Response link will open a new window containing the SOAP response that was returned. You can use this information to determine why the stored procedure failed to execute properly.

If the stored procedure is executed successfully, you will be taken to the finalize panel, where SOLA will display information about the parameters used by the stored procedure.



The information is presented under a series of column headings:

- **Level:** the parameter's logical level.

- **Parameter Name:** the parameter name.
- **Input/Output:** the parameter type, either I (input), O (output) or B (both).
- **Length:** the parameter's internal length.
- **Scale:** for some types of parameters (such as fractions), the scale represents the number of significant positions after the decimal point.
- **Occurs:** if the parameter is an array or a table, this indicates how many times it occurs.

You can view the SOAP request generated by SOLA and the response returned by the stored procedure.

To view the SOAP request, click the request link. Likewise, click the response link to view the response.

If the procedure executed correctly and the request and response are satisfactory, you are ready to finalize the registration.

Click FINALIZE to create the web service.

Using SOLA Developer – Ad-hoc SQL

SOLA provides the means to register a method within your project that will let you run Adhoc SQL as a web service.

How SOLA Creates Web Services from Ad-hoc SQL

To run Adhoc SQL as a web service, SOLA creates a dummy program that will represent your Adhoc SQL requests. You do not need to register each SQL request as a separate method. One dummy program will provide access to all of your Adhoc SQL.

Creating a Web Service from Ad-Hoc SQL

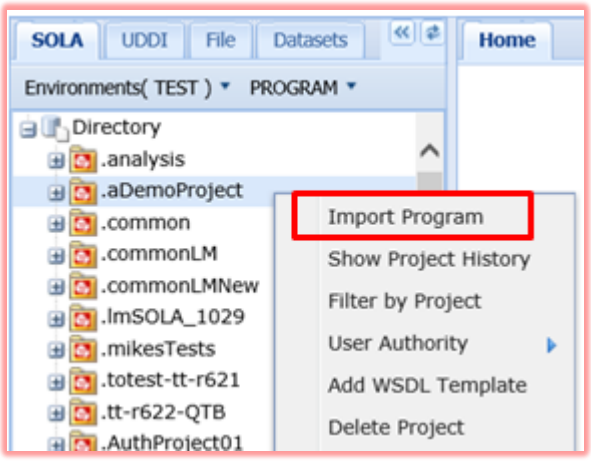
This section will describe the steps necessary to register a method to run Adhoc SQL as a web service. There is no analysis when making web services from Adhoc SQL. The end result will be a WSDL, metadata template, test harness and a UDDI entry.

Step 1 – Mainframe Preparation

To configure the mainframe environment, you will need to set up PPT entries in the SOLA WOR.

WOR	**AOR**
PPT: DEFINE PROGRAM(DummyprogName) LANG (LE) STATUS (Enabled) REMOTE REGION(yourRegion) REMOTE NAME:XMLPC200 REMOTE TRANID: (yourTranId)	**PPT:** DEFINE PROGRAM(XMLPC200) LANG (LE) STATUS (Enabled)
	PCT: DEFINE TRANSACTION(yourTranId) PROGRAM (DFHMIRS) PROFILE (DFHCICSA) STATUS (Enabled)

Step 2 - Adhoc SQL Registration

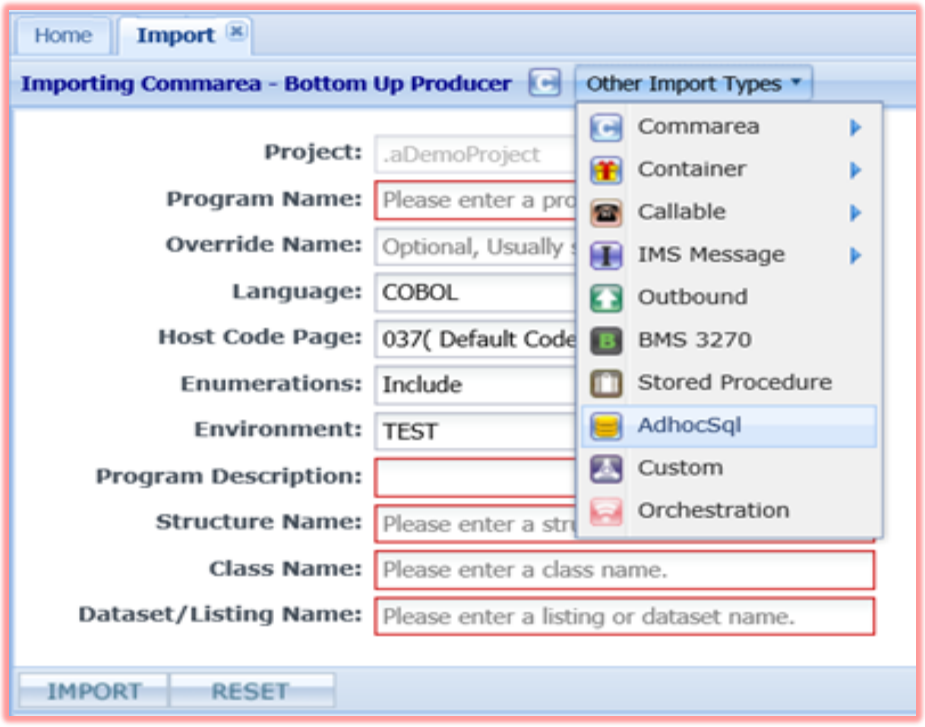


Select the project you wish to import to and right-click it. From the pop-up menu, select **Import Program**. If you wish to create the program in a new project, first follow the steps for creating a new project on page 22.

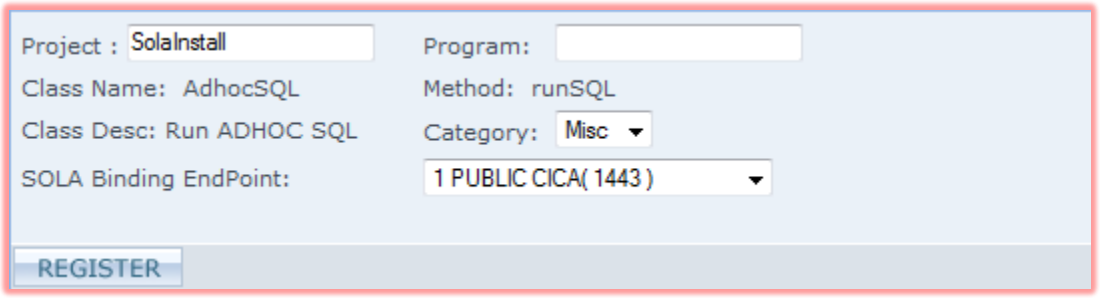
In order to import a program into a project, you must be an authorized user of that project. Once the program is imported, you can drag and drop the program from one project to another. However, you must also be an authorized user of the project you wish to move the program into.

After you select **Import Program**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

The default program type is commarea, so use the **Other Import Types** menu to select Adhoc SQL.

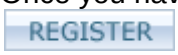


The Import panel will change to display the Adhoc SQL panel.



The only information you need to provide is the program name and the binding endpoint.

- **Project:** the name of the project to which the dummy program will belong. This is pre-populated based on the project from which you accessed the import screen and cannot be changed.
- **Program:** the program name that will represent your Adhoc SQL requests. As this is not an actual program, you may enter any unique name up to eight characters long.
- **Method:** this will be the method you use to represent your Adhoc SQL requests. It's default name, runSQL, cannot be changed.
- **Class Name:** the class name that will represent the dummy program. Class names are necessary for the dummy program to be used in distributed systems. It's default name, AdhocSQL, cannot be changed
- **Class Description:** a brief description of the dummy program. This cannot be changed.
- **Category:** the category (type) of program. Options vary by installation.
- **SOLA Binding Endpoint:** the mainframe endpoint in which the dummy program will run.

Once you have entered the program name, optional category and a binding endpoint, click 

to create the dummy program.

Using SOLA Developer – Custom Programs

Unlike most legacy CICS programs, such as COMMAREA programs, Custom programs are written for the specific purpose of hosting web services within CICS using SOLA. Custom programs allow for the complete control and handling of both the SOAP request and the SOAP response or SOAP Fault. That is to say that all interrogation of the SOAP request, and building of a valid SOAP response, is handled by the custom program using the SOLA provided DOM API. In addition Custom programs are not subject to the 32K data limit typically imposed on COMMAREA programs.

There are currently three distinct varieties of Custom programs supported by SOLA.

1. Version one 'Legacy' (the SOLA 5.1 model). These are typically programs written using SOLA 5.1. These programs must be written using the SOLA 5.1 DOM API. Although transparent to the programmer, SOAP requests and responses over 32K in size are communicated between WOR and AOR regions by use of a SOLA provided DB2 table called the 'overflow' table.
2. Version one 'SOLA 6.1 model'. A newer, SOLA 6.1 provided version works in mostly the same manner as the Legacy version except that it uses the new SOLA 6.0 DOM API. **Note:** This version of Custom programs was introduced in SOLA 6.1 PTF SFX-6116 and IDE Release Version 6.1.10 and greater. The procedures described here reflect this new model. Legacy custom programs will still be supported, but any **new development should comply with the new standards described here**.
3. Version two 'Container model'. This SOLA 6.1 supported version is completely different from the others in that it is based on CICSs Containers. In this case the programmer has to do a few more steps compared to the other versions. The SOAP request must first be retrieved from a provided CICS Container, and the SOAP Response (or Fault) must be placed on a CICS container to be passed back to the WOR region. The advantages of this model are many. It has less architectural complexity in that it does not require the use of DB2. It also means that the Custom program may use ANY parser desired (does not have to be SOLA's Parser/DOM API) or no parser at all. In addition the required PPT entries are more straightforward.

Since the 5.1 version of the DOM API is deprecated (no longer supported) the version of Custom program that uses it is also deprecated. Therefore, for the purposes of this document, only the newer variety of Version one (the SOLA 6.0 model) will be discussed along with Version two.

The process for creating a web service from a DOM API (Custom) program is different than other program types in that, instead of having an existing program which you expose as a web service and then test, you will first write the Custom program code, test it, and then import it into SOLA for registration and the building of the WSDL and test harness etc.

Custom Programs Version 1 (using the SOLA 6.1 DOM API)

With this version, SOLA program XMLPC202 will accept the SOAP Request passed to it from the SOLA Listening Region and parse the request using SOLA's 6.1 DOM Parser. The DOM tree handle created by parsing will be passed via a CICS Link to your Custom Program. The DFHCOMMAREA that your custom program receives is described in a SOLA provided copybook called XMLCUV12.

Description of Copybook XMLCUV12:

```
05  :LK-:DOM-HANDLE           USAGE IS POINTER.
05  :LK-:VER-EYE-CATCHER      PIC  X(07).
05  :LK-:SOAP-REQUEST-SZ      PIC S9(09) BINARY.
05  :LK-:CUST-RETURN-CD       PIC S9(04) BINARY.
    88  THROW-SOAP-FAULT      VALUE -1.
05  :LK-:FAULT-DETAILS.
    10  :LK-:FAULT-CODE       PIC  X(01).
        88  SERVER-ERROR      VALUE 'S'.
        88  CLIENT-ERROR      VALUE 'C'.
    10  :LK-:FAULT-STRING     PIC  X(35).
    10  :LK-:FAULT-MSG        PIC  X(254).
```

Note that when copying this into your program you need to use a copy format such as:

```
COPY XMLCUV12 REPLACING ==:LK-:== BY ==LK-==.
```

LK-DOM-HANDLE – This is the handle that represents the DOM tree created as a result of parsing the SOAP request.

LK-VER-EYE-CATCHER – Should be valued: 'CU01.02'.

LK-SOAP-REQUEST-SZ – The size of the SOAP Request.

LK-CUST-RETURN-CD – Indicates to SOLA the action to be performed when your Custom program is completed. When zero, SOLA will use the LK-DOM-HANDLE (which should be the DOM Handle your program used to create the SOAP Response) to generate the SOAP response to send back to the service requestor.

LK-FAULT-DETAILS – If LK-CUST-RETURN-CD is set to -1 then SOLA will use the information In this data structure to format and throw a SOAP Fault.

The manipulating of SOAP request and response must be handled by the SOLA 6.0 provided DOM API. The SOLA DOM API can be used by CICS transactions and Batch jobs, while the SOLA DOM parser and API together provide functions to inquire on an XML document repeatedly in any direction, as well as a method to create new XML documents from COBOL programs.

Since parsing of the SOAP request is handled by XMLPC202, programs of this type need only interrogate the request using the DOM API and build a proper SOAP response, also using the DOM API.

When finished building the response the program only needs to ensure that the DOM handle used to create the response is placed into the commarea field LK-DOM-HANDLE and return control to XMLPC202. XMLPC202 will then take care of delivery of the response to the requestor.

Custom Programs Version 2 - Containers

This style of Custom program requires the use of CICS containers. Unlike Version 1 programs which transfer SOAP data larger than 32K using a DB2 table called the 'overflow' table, this version transfers the SOAP data (regardless of size) in a CICS container. This version requires no SOLA program in the AOR (XMLPC202 is required in version 1) and control is, therefore, passed directly to your custom program from the WOR region. This means that your program will be responsible for the proper handling of the various containers passed between the WOR and the AOR. In addition your program will need to parse the SOAP request that is passed in the 'request' container, and place a valid soap response in the 'response' container along with control information is the 'status' container. Although a bit more work will be required from your program to do this (as compared to a version 1 program) the advantages are a simplified architecture, no DB2 requirement, and the flexibility of using any parser, or parsing technique you like.

The following is a list of possible containers that your custom program may use (all containers are contained within channel SOLA-CUSTOM):

- **SOLA-STATUS:** Communications pertaining to status are handled through this container. Contains SOLACUV2 copybook is described later in this section. This container is input/output.
- **SOAP-REQUEST:** Contains the decrypted SOAP request. This container is input only.
- **SOAP-RESPONSE:** This container is optional. It should be present only if a normal return code is present in the status container and will contain your SOAP response. This container is output only.
- **SOAP-FAULT:** This container is optional. If you want to create a custom fault, you should put that fault in this container. This container is output only.

Once the SOAP request has been retrieved, your program can use any parser to process the request and build either a SOAP response or a fault (we recommend the SOLA 6.0 parser of course).

Once the SOAP response or fault has been built, it will need to be placed back into a specific container (along with the proper status). When control returns to the SOLA region, SOLA will interpret the status and take appropriate action (i.e. under normal circumstances, it will send back the SOAP response).

Under normal circumstances you will place the completed SOAP response into the CU-RESP-CONTAINER container and set the CU-RETURN-CD to zero. In this case SOLA will simply deliver your soap response to the client requestor.

Required Copybooks

The copybook SOLACUV2 is passed to the custom program in the status container. You will need to retrieve it into the custom program, update the information in the data area and place it back into the status container. This is used to report status information from the custom program and maps the area passed to the application linkage.

Note: When SOLA 6.0 PTF SFX-6116 and IDE Release Version 6.1.10 are applied, you will also get following copybook updates as a part of SAMPLIB:

XMLDOMW1 - Interface for new DOM API (We have added new 88 level items so the flags match old interface).
XMLCUV12 - Copybook that maps the area passed to Application linkage

The following are the contents of the SOLACUV2 copybook:

```
05  CU-RETURN-CD            PIC S9(04) BINARY.
    88  CU-RETURN-NORMAL    VALUE +0.
    88  CU-THROW-FAULT      VALUE -1.
    88  CU-CUSTOM-FAULT     VALUE -2.
05  CU-RETURN-MSG           PIC  X(100).
05  CU-CHANNEL-NM           PIC  X(16) VALUE 'SOLA-CUSTOM'.
05  CU-STATUS-CONTAINER     PIC  X(16) VALUE 'SOLA-STATUS'.
05  CU-STATUS-LEN           PIC S9(09) BINARY.
05  CU-REQ-CONTAINER        PIC  X(16) VALUE 'SOAP-REQUEST'.
05  CU-REQUEST-LEN          PIC S9(09) BINARY.
05  CU-RESP-CONTAINER       PIC  X(16) VALUE 'SOAP-RESPONSE'.
05  CU-RESPONSE-LEN         PIC S9(09) BINARY.
05  CU-FAULT-CONTAINER      PIC  X(16) VALUE 'SOAP-FAULT'.
05  CU-FAULT-LEN            PIC S9(09) BINARY.
```

The names of the SOAP-RESPONSE and SOAP-FAULT containers can be overridden by your custom program if you wish. Other container names need to remain the same.

Faults

You can throw two types of faults. If you set the CU-RETURN-CD to -1 SOLA will throw the message contained in the CU-RETURN-MSG area as a SOAP Fault (your program need not create an actual SOAP fault). You can alternately set the CU-RETURN-CD to -2 and SOLA will retrieve your custom SOAP Fault from the CU-FAULT-CONTAINER and send it to the requestor.

Sample Program

For a sample custom program, see Appendix D.

Creating a Web Service from a Custom Program

This section will describe the steps necessary to create a web service from a DOM API program.

Step 1 – Mainframe Preparation/Coding Custom Program

PPT Entries

To work with SOLA, DOM API programs require a PPT entry in the WOR region that points to the AOR region. Prior to SOLA 6.1 PTF SFX-6116 and IDE Release Version 6.1.10 the remote PPT definition for custom program invokes XMLPC200 on AOR region. This is the SOLA 5.1 model and should continue to be used for support those legacy programs. To exploit the **new DOM API interface** (SOLA 6.1 model) the remote PPT definition has to be setup to invoke **XMLPC202**. If using the Container model the remote PPT definition should invoke the actual Custom program (as with most common DPL entries).

WOR	**AOR**
PPT Entry	**PPT:**
Version 1:	DEFINE PROGRAM(XMLPC202)
DEFINE PROGRAM(program name)	LANG (LE)
LANG (LE)	STATUS (Enabled)
STATUS (Enabled)	
REMOTE REGION(yourRegion)	**PCT:**
REMOTE NAME (XMLPC202)	DEFINE TRANSACTION(yourTranId)
REMOTE TRANID (yourTranId)	PROGRAM (DFHMIRS)
Version 2: (using containers)	PROFILE (DFHCICSA)
DEFINE PROGRAM(program name)	STATUS (Enabled)
LANG (LE)	
STATUS (Enabled)	
REMOTE REGION(yourRegion)	
REMOTE NAME (yourProgramName)	
REMOTE TRANID (yourTranId)	

Step 2 - Testing the Program

It is recommended that you test the program using the SOLA Raw Tester before creating a web service. For instructional purpose, you can use the sample program "Convmph" that is shipped with SOLA. This sample program uses the DOM API to consume and create XML, and also uses the XML format conversion program to convert data into and out of XML string notation. It has one simple function, to convert miles per hour into kilometers per hour. The program accepts a single value as input and creates a single value as output.

Convmph demonstrates using the DOM API to retrieve a value, convert the value, use it in a calculation, create an output XML document and include the output value in the document. Sending a SOAP fault is also demonstrated.

Convmph expects an input XML message like the one below:

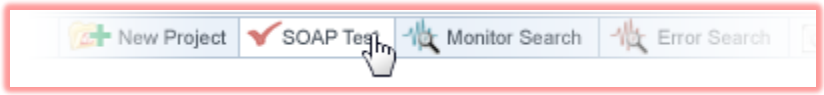
```
<mph>value</mph>
```

Before executing the XML input message, wrap it in a SOAP message as follows.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ConvMph xmlns="http://convMPH..SOLAsoa.com/CU/CONVMPH/ ">
          <mph>value</mph>
        </ConvMph>
    </soap:Body>
</soap:Envelope>
```

The namespace must specify "CU" followed by the program name

Access the raw tester by clicking the **SOAP Test** button on the button bar. This will display the raw test panel.



Enter the request shown above into the test field, using a value of 100 as input.

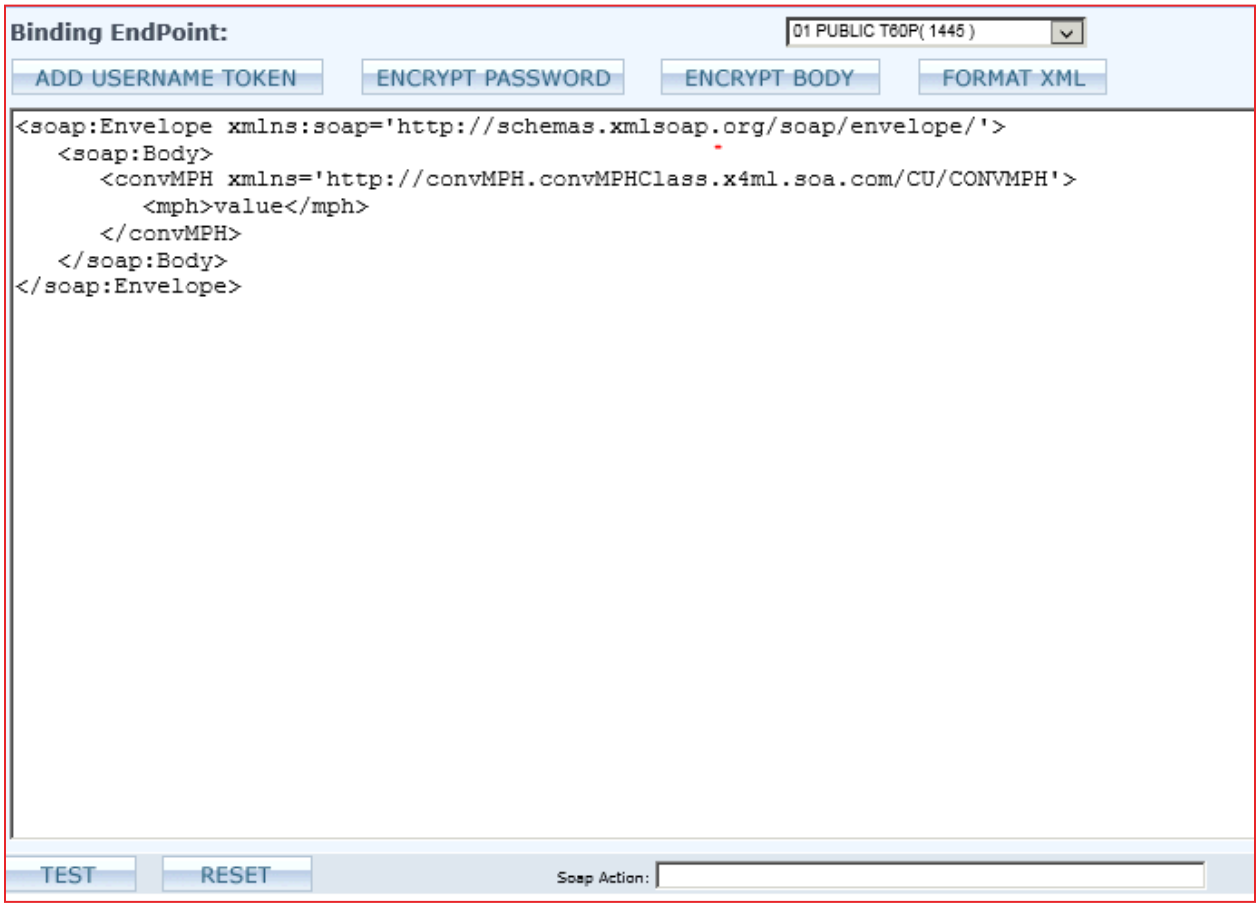
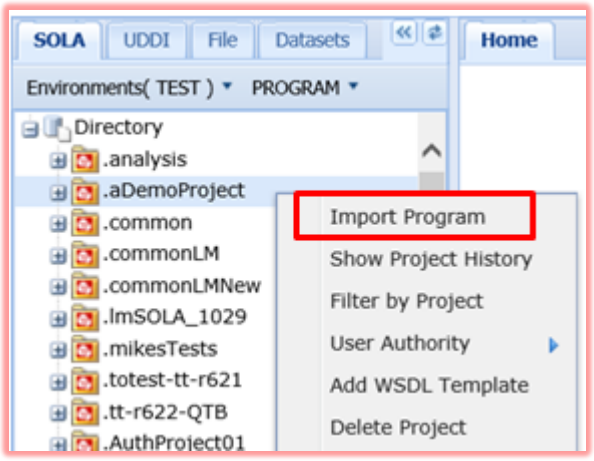


Figure 1 – Test SOAP Request

Click  to test the program.

Assuming everything is set up correctly, Convmph will return a SOAP response (in a new browser window), in which 100 mph has been correctly converted to 160.93 kph.

Step 3 - Import Custom Program

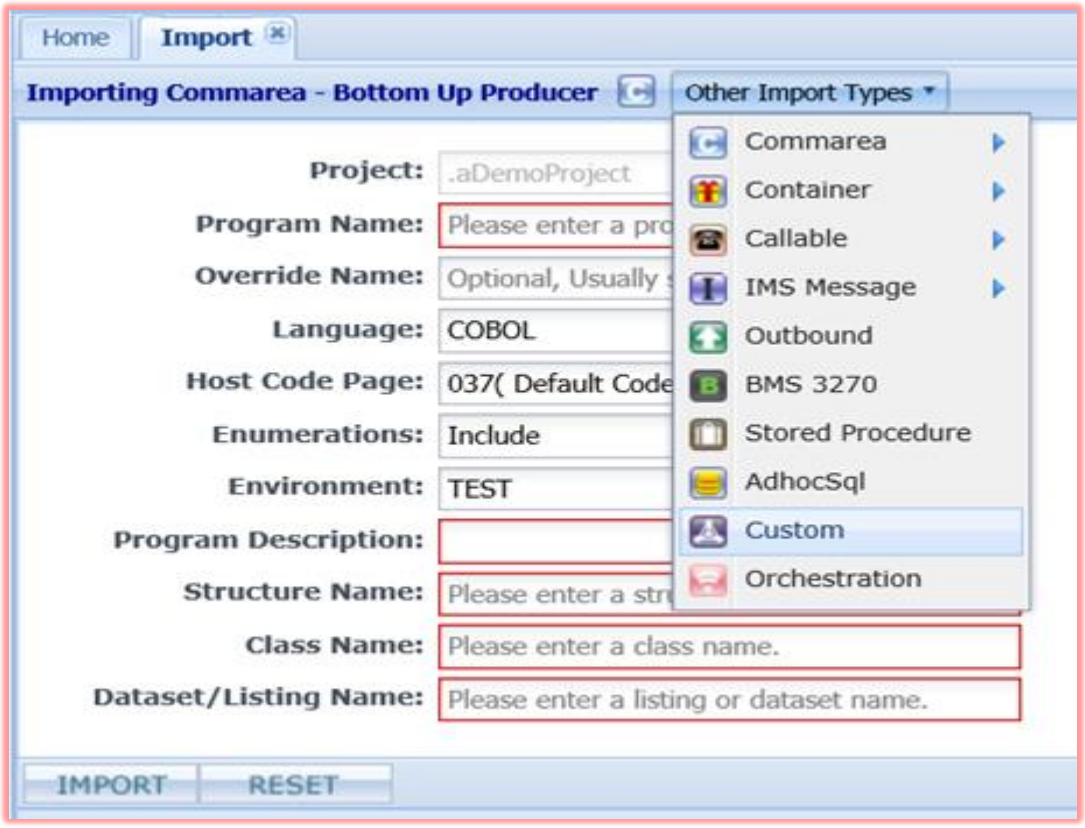


Select the project you wish to import to and right-click it. From the pop-up menu, select **Import Program**. If you wish to create the program in a new project, first follow the steps for creating a new project on page 22.

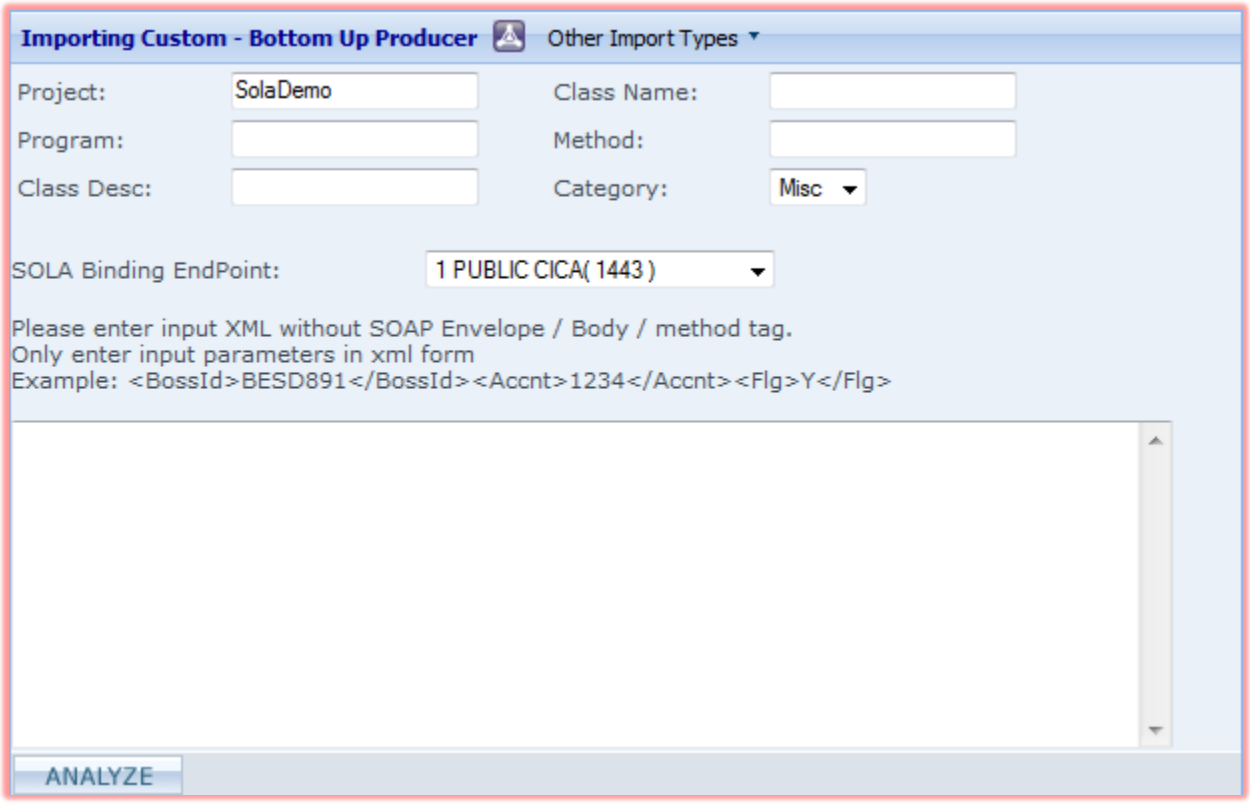
In order to import a program into a project, you must be an authorized user of that project. Once the program is imported, you can drag and drop the program from one project to another. However, you must also be an authorized user of the project you wish to move the program into.

After you select **Import Program**, the Import panel will be displayed under a tab in the workspace. This panel can be used to import any program type that SOLA supports.

The default program type is commarea, so use the **Other Import Types** menu to select Custom.



The Import panel will change to display the Custom Program import panel.



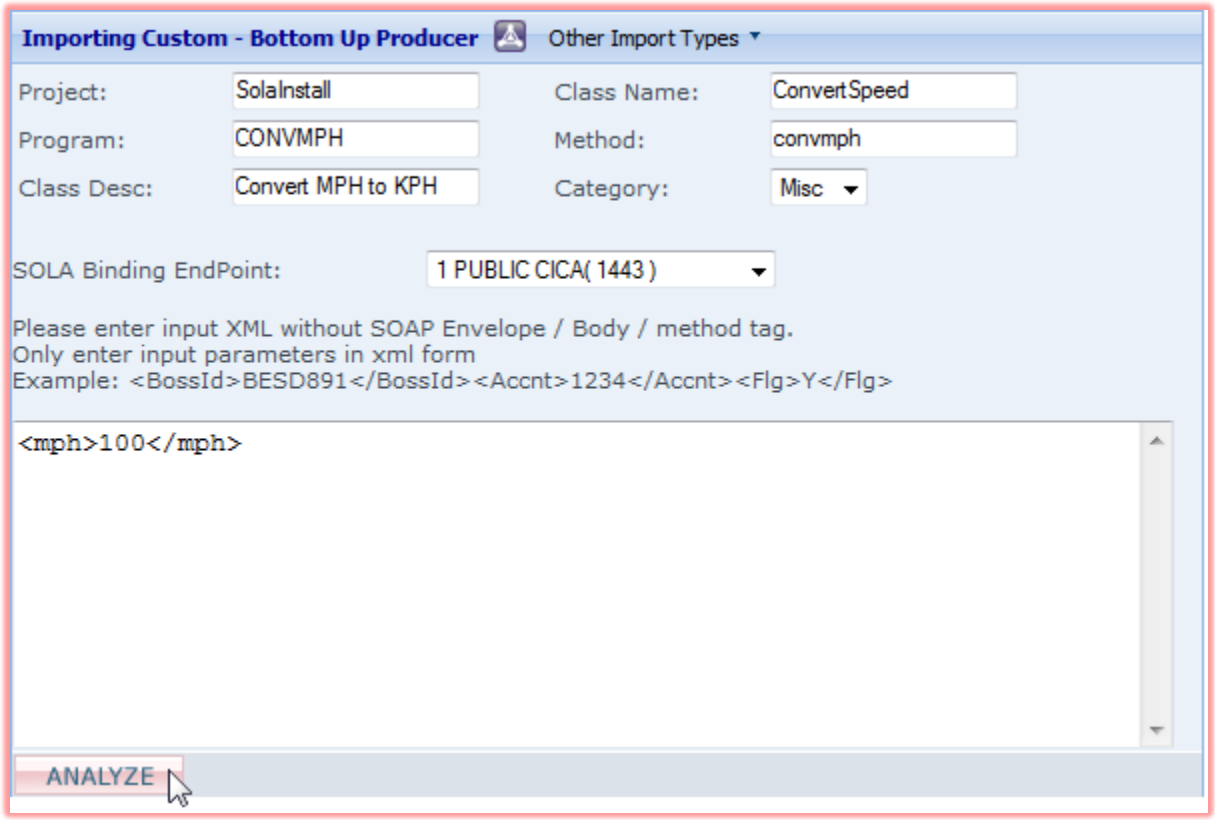
The Import panel consists of a series of fields used to provide information about the source program and the destination SOLA program that will be created.

- **Project:** the name of the project to which the imported methods/operation will belong. This is pre-populated based on the project from which you accessed the import screen and cannot be changed.
- **Program:** the name of the program to which the new method will belong. Unlike bottom up commarea analysis, you cannot create a program on its own.
- **Method:** the name of the method/operation to be created.
- **Class Name:** when you create a web service from a DOM API program, it will be exposed as a method. Distributed systems classify methods as belonging to a class. Therefore, SOLA requires that you assign a class name to the program to which this method belongs.
- **Class Description:** a brief description of the program.
- **Input field:** this is the large empty field towards the bottom of the panel. This is where you insert the XML input that the DOM API program requires to run. Just as with other program types, SOLA requires a valid sample input value to run the program. When the web service you create is published, consumers will be able to submit any value they

chose. When entering the input, use XML format without a SOAP envelope, body or method tags.

- **Category:** the category (type) of program. Options vary by installation.
- **SOLA Binding Endpoint:** the mainframe endpoint in which the program will run.

When you have filled in all of the required information, click ANALYZE .



Custom programs use the standard (bottom up) commarea analyzer, and analysis is performed the same way.

Test Harness

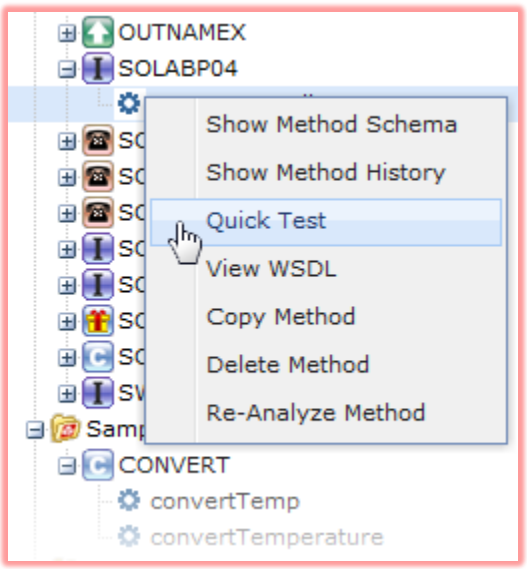
SOLA Developer uses two testing tools to test web services; the quick tester and the raw tester.

- **Quick Tester:** used to test a specific method. The quick tester runs the web service and asks for the inputs the web service requires, then makes a soap call and provides the SOAP response as raw XML. The tester also provides comprehensive configuration capabilities, allowing the user the make changes to xml structure for debugging purposes. It is considered good practice to test every web service that you create.

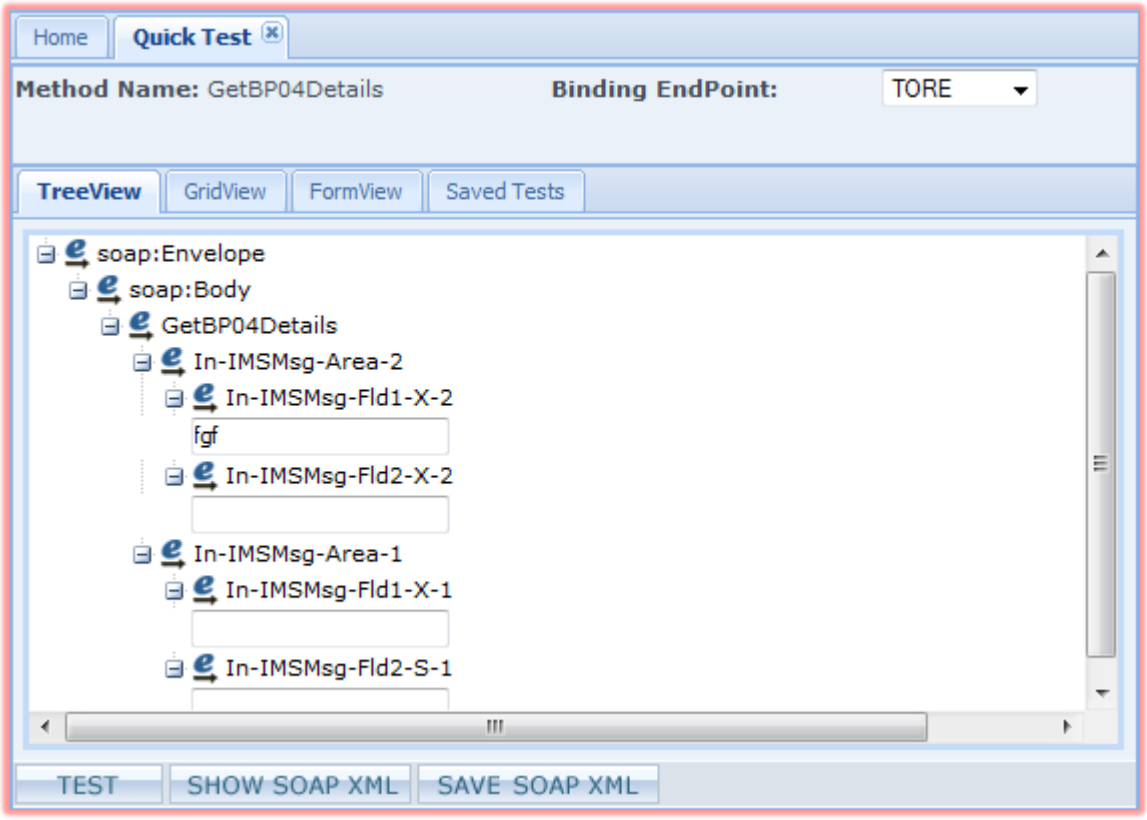
- **Raw Tester: used to test raw XML.** Users can copy and paste XML (SOAP Request, etc.) into the Raw Tester, edit it as necessary, and send it as a SOAP request. SOLA will then query the target legacy program and send a SOAP response as raw XML. This testing facility is very useful for testing customizations and tweaks in situations where the user either cannot or does not want to make changes using one of the SOLA analyzers.

Quick Tester

To access the quick tester, click on the method you want to test and select "Quick Test" from the pop-up menu. This will display the quick test panel.



The purpose of the quick test panel is to display the method's required inputs, have the user enter values for those inputs and then submit the SOAP request to the legacy application, just as the web service would if it were in production. The user can also make changes to the XML structure for debugging purposes, using either drag and drop functionality or manually editing the XML. In this manner, you can experiment with the web service, figure out what it needs to make it work, and then go back to analysis and make those changes. The Quick Tester supports HTTPS.



The quick test panel has three views, any one of which can be used to enter values for the required inputs. There is also a saved tests view, detailed below.

- **Tree View:** this is the default view and is a compromise between simplicity and configurability. The inputs are clearly displayed and easily configured and there is a

certain amount of customization you can do with the XML structure by dragging and dropping tree items.

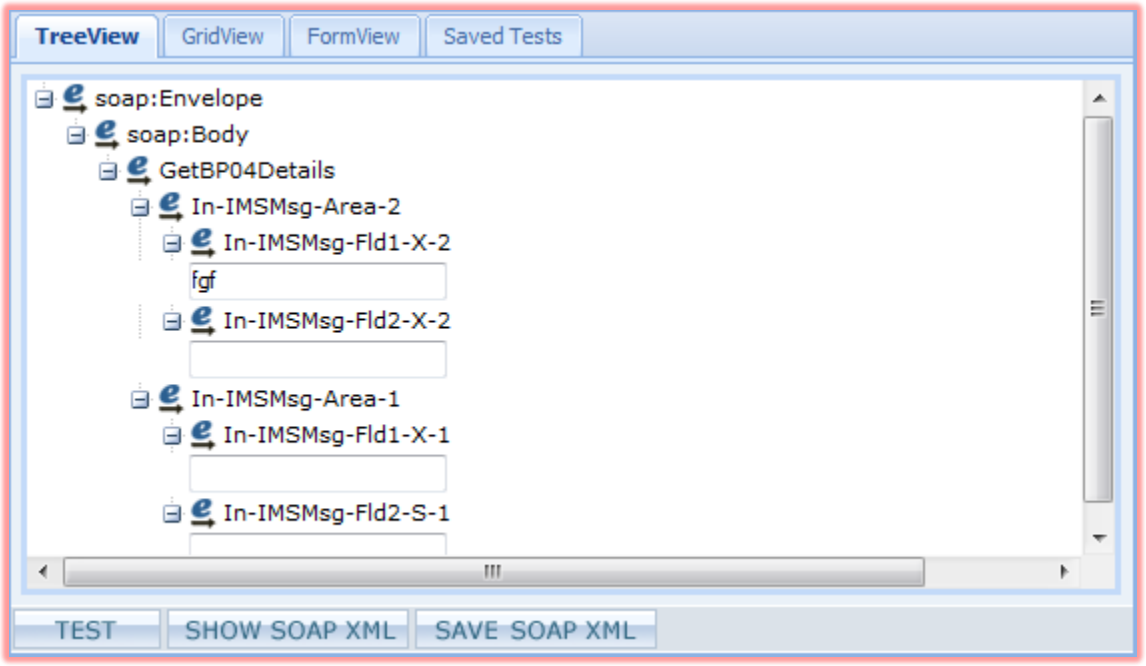
- **Grid View:** this is the simplest to use but least configurable view. You cannot change anything, but the inputs are all laid out in a neat table for ease of use.

- **Form View:** this is the least user friendly but most configurable view. The SOAP request is shown as raw xml. To make changes to the inputs, you have to change the XML manually. This allows for tremendous customization (you can change the XML however you like) but is not very easy to use.

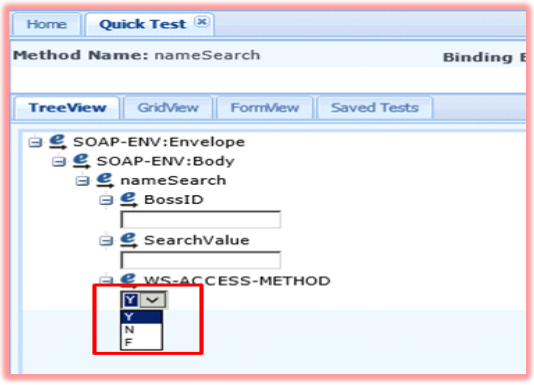
- **Saved Tests:** when configuring a test, you can use the button SAVE SOAP XML to save the information you entered. The Saved Tests tab contains a list of saved tests, and clicking on a test in this list will restore the saved configuration information. You can also delete saved tests using the icon.

Tree View

The tree view is the default view and is a compromise between simplicity and configurability.

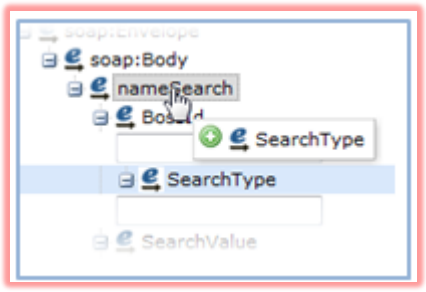


The first and default view (shown above) displays the same XML structure tree that you saw in the commarea analyzer, though it only shows the input half. For every tree item that was described in analysis as a variable input (an input without a fixed value) will have either a text box (if there were no enumerations/restrictions) or a drop down menu (if there were enumerations).



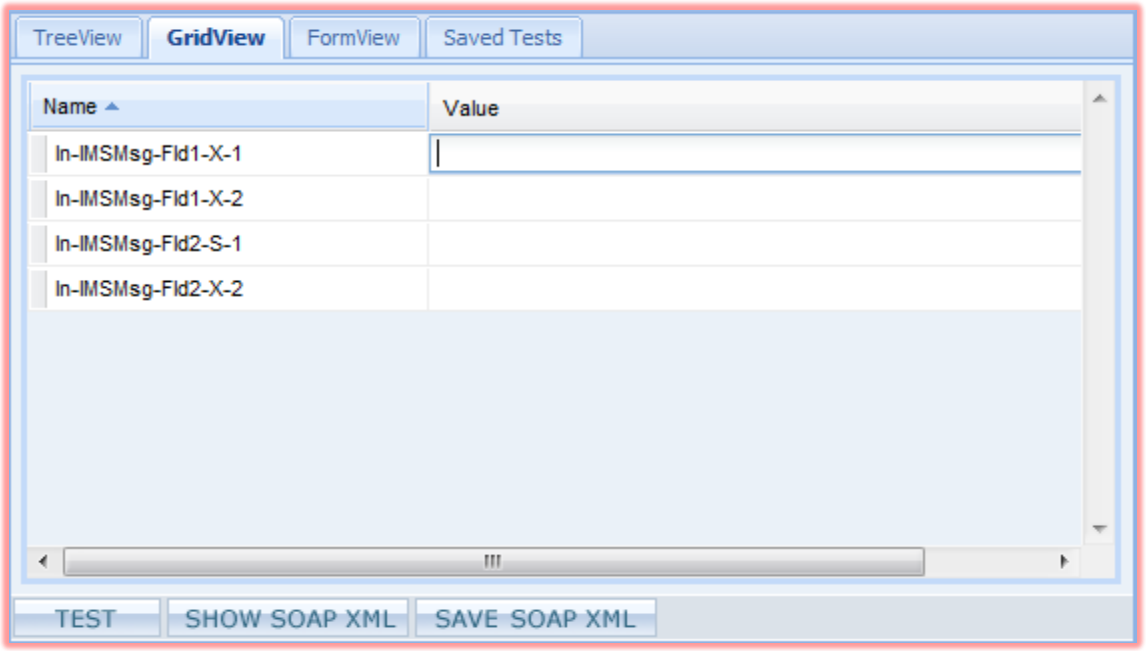
QuickTest of methods having enumerated schema items is now displayed in a drop-down box with valid enumerated values for selection by user.

You can also drag and drop items from one position in the tree to another to experiment with XML structure (as seen in the illustration on the right). Keep in mind, however, that if you discover a fault in your original structure and can successfully execute the web service with a new structure you've created in the quick test panel, you will have to go back to analysis and make the same changes there.



Grid View

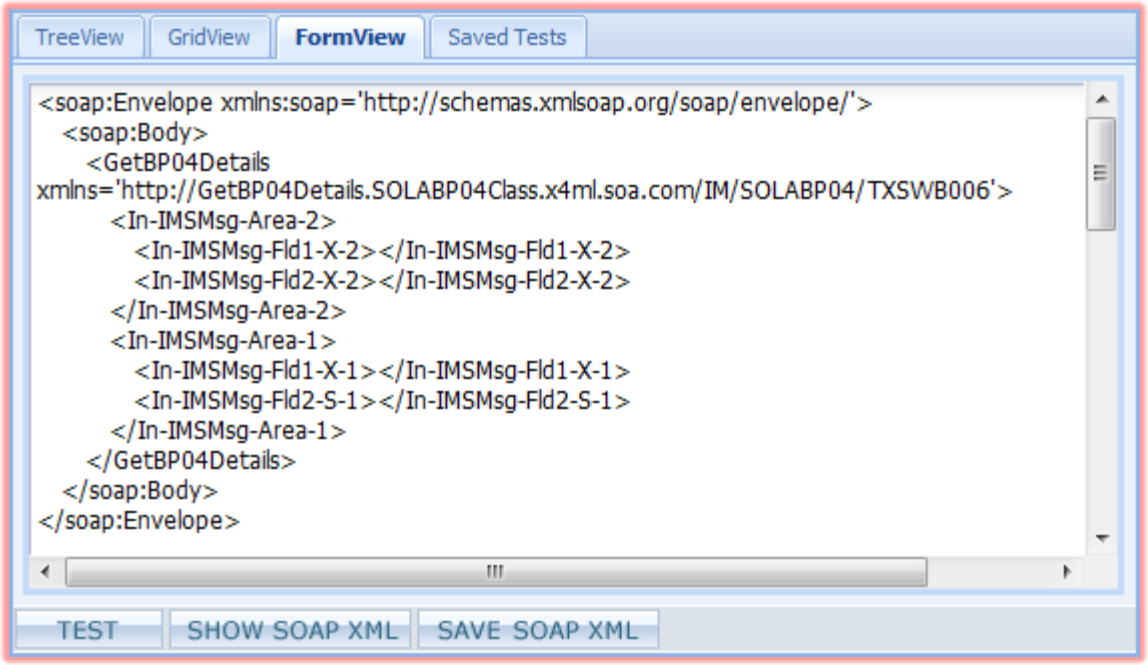
The second view is the grid view, which shows all of the inputs as either text boxes or drop down menus, but does not show the XML structure or allow you to tweak the positioning of the tree items.



This is the simplest and least configurable view mode to use. The only thing you have to do is enter the required inputs.

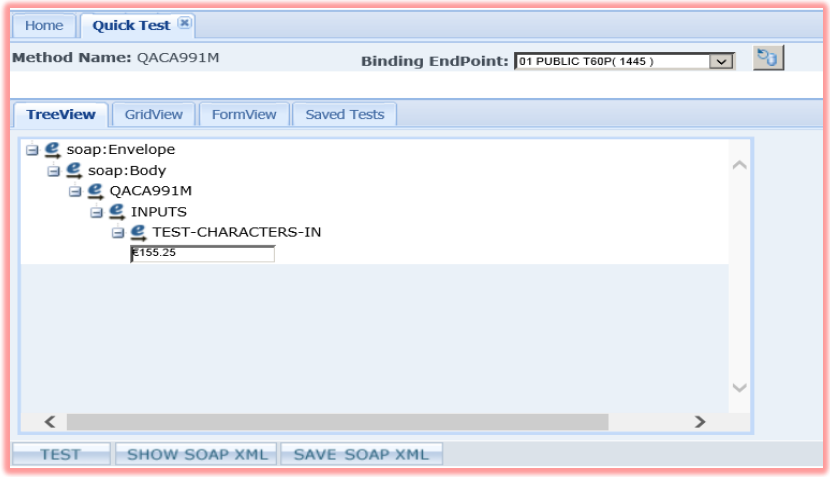
Form View

The third and last view is the form view, which displays the SOAP request as raw XML. This is the most configurable view as you can make whatever changes you want to the XML directly. In many ways, this is like the raw test panel (described later in this chapter), but it is pre-populated with the method's SOAP request, minus the user configurable variables.



To use this view, enter the inputs directly into the XML and make whatever changes you need in case the web service doesn't work. Using this view requires an understanding of XML.

Testing the Method



Once you have configured the inputs using one of the three view types, click TEST to send the SOAP request.

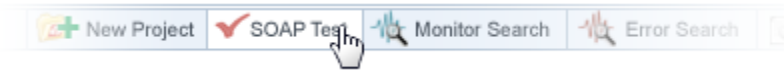
You can also view the SOAP request that the web service will send to the legacy application by clicking SHOW SOAP XML.

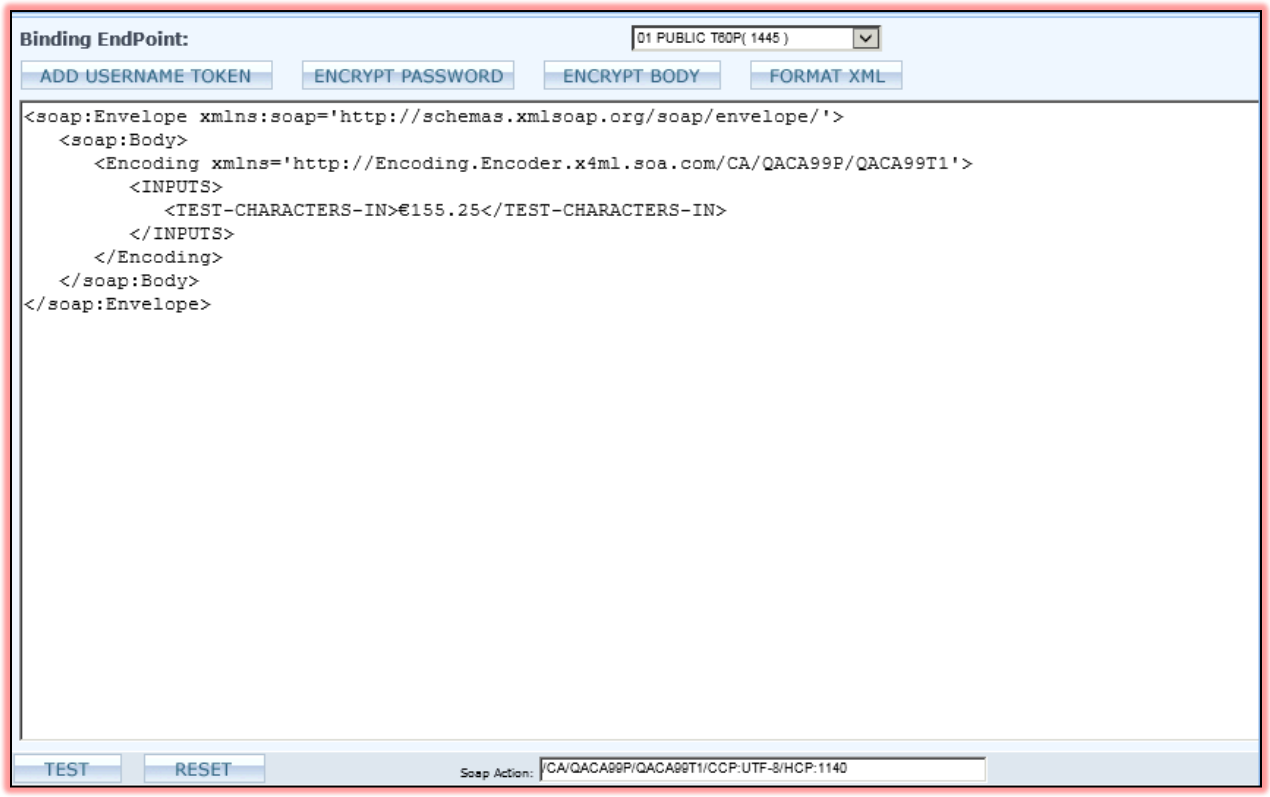
The **response** will be returned as a SOAP response in a new browser window (or tab) as seen here:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- StartTime( 2014-06-28-12.48.51.000292 ) EndTime( 2014-06-28-12.48.51.000355 ) ElapsedTime( 63 milliseconds )
-->
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  - <Body>
      - <QACA991MResponse
        xmlns="http://QACA991M.CLASS_QACA991P.x4ml.soa.com/CA/QACA99P/QACA991T">
          - <OUTPUTS>
              <TEST-CHARACTERS-OUT>€155.25</TEST-CHARACTERS-OUT>
            </OUTPUTS>
        </QACA991MResponse>
    </Body>
</Envelope>
```

Raw Tester

To access the raw tester, click the **SOAP Test** button on the button bar. This will display the raw test panel.





The Raw Test screen is used to test a piece of SOAP code in its raw state (i.e. SOAP code can either be manually entered or pasted for testing). Tests initiated from this screen use http as a transport. The Raw Tester does not support HTTPS.

The raw tester has options not available in the quick tester:

Click this button to add a WS-Security header to your SOAP message, with your mainframe UserId and password in the WS-Security header.

Click this button to encrypt the password text of Username Token

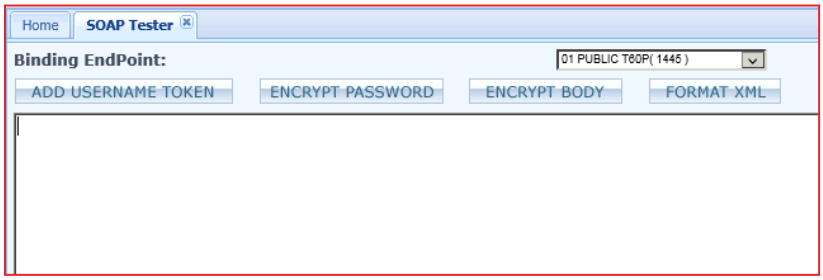
Click this button to encrypt the entire body of the SOAP XML.

Click this button to indent and make the XML more readable.

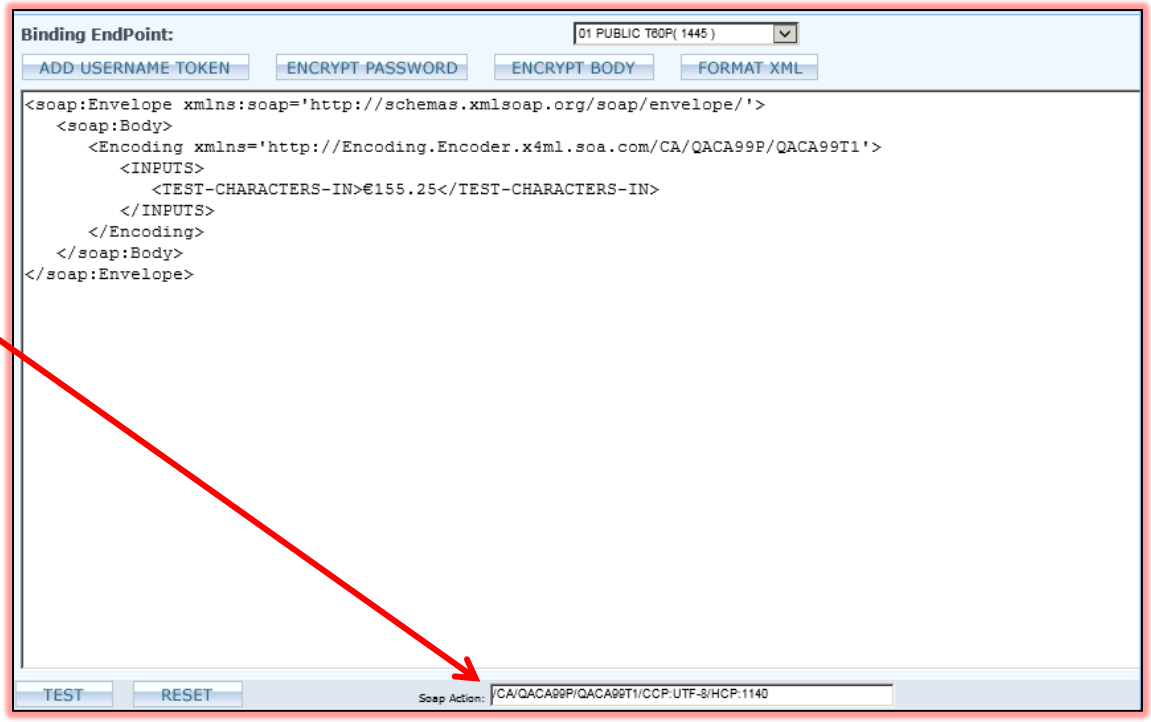
To test a piece of SOAP code, Click on 

and the **SOAP Tester** tab will be presented with the large text box in the workspace. either paste it into the large text box or manually enter it. Click 

to send the SOAP call.

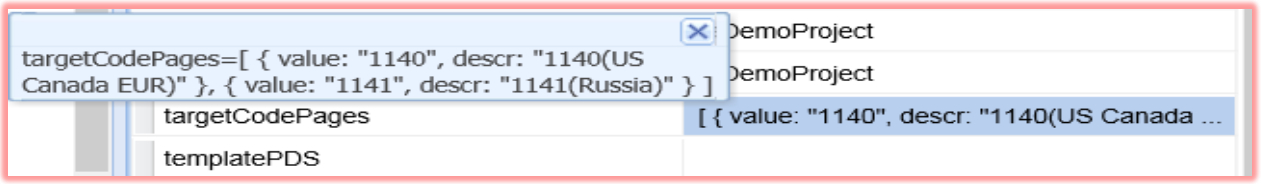


Note: Soap Action is required if using a Host Code Page other than the default EBCDIC Host Code Page 37.



Host Code Pages are defined and setup in the Property File located in the **codepages xml** file; these are defined with a property name of 'targetCodePages'.

Right clicking on the Project Name in the Directory Tree will list all properties associated with the Project. Scroll down the list and you will see all code pages that have been defined under property name 'targetCodePages' if other than the default code page 37 is being used.



Monitoring and Logging

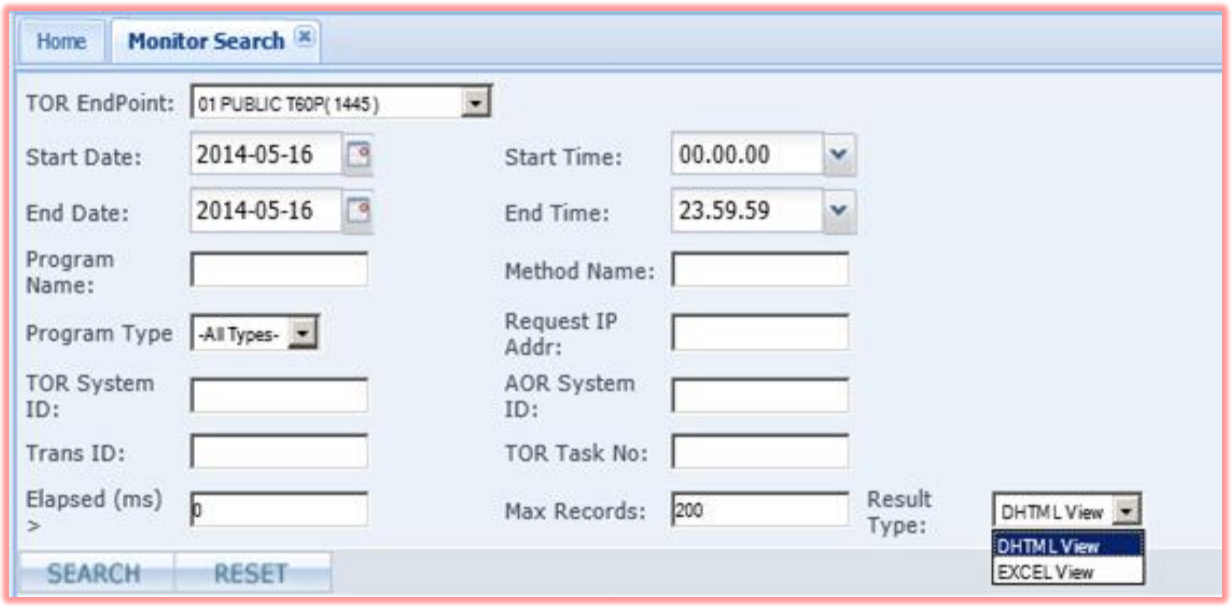
SOLA offers a complete set of monitoring, logging and error reporting tools. The SOLA run-time automatically logs every transaction in the SOLA Monitor log, which is designed to handle high-volume transactions while not adding overhead to a transaction. The run-time does this by logging transactions to an in-memory structure (a CICS user-maintained data table (UMT)). Background transactions spool this data from the UMT to a DB2 table.

Any time the SOLA run-time detects an error (SOAP fault, program abend, parsing error, etc.) it logs a message in the SOLA Error log. The SOLA Error log is written directly to a DB2 table. It contains the full input SOAP message, the error message and a link to the transaction in the SOLA Monitor log.

Transaction Logs

You can search the transaction logs by clicking the **Monitor Search** button on the button bar.

This will display the monitor search panel.



To conduct a search of the transaction log, enter search parameters using the search fields to narrow the scope of your search. You can also conduct a search with the default (mostly blank) settings, though this may take some time to complete and may result in a very long list of transactions.

The following is a description of the search fields:

- **TOR EndPoint:** narrows the search to transactions within a matching TOR region.

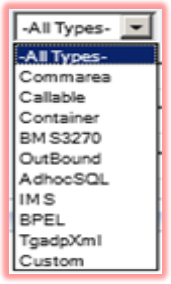
- **Start Date and End Date:** the start and end dates are automatically populated with the current date, though these values can be changed if necessary. All transactions are stamped with the date and time at which they take place and only transactions that took place on or after the start date and on or before the end date will be returned.

- **Start Time and End Time:** the start and end times are automatically populated with the current system time and can be changed by manually entering a time (hh.mm.ss). All transactions are stamped with the date and time at which they take place, and only transactions that took place at or after the start time and at or before the end time will be returned.

- **Program Name:** narrows the search to transactions executing this program.

- **Method Name:** narrows the search to transactions generated by the execution of the specified method.

- **Program Type:** narrows the search to transactions initiated by a method executed by the specified program type. Options are **All Types** listed in the illustration to the right:



- **Request IP Addr:** narrows the search to transactions generated in response to a request that originated from an IP address matches the specified IP address (if the request came via HTTP).

- **TOR System ID:** narrows the search to transactions with a matching TOR system Id.

- **AOR System ID:** narrows the search to transactions with a matching AOR system Id.

- **Trans ID:** narrows the search to transactions with a matching transaction Id.

- **TOR Task No:** unique identifier that is given to each unique instance of a program running in a TOR.

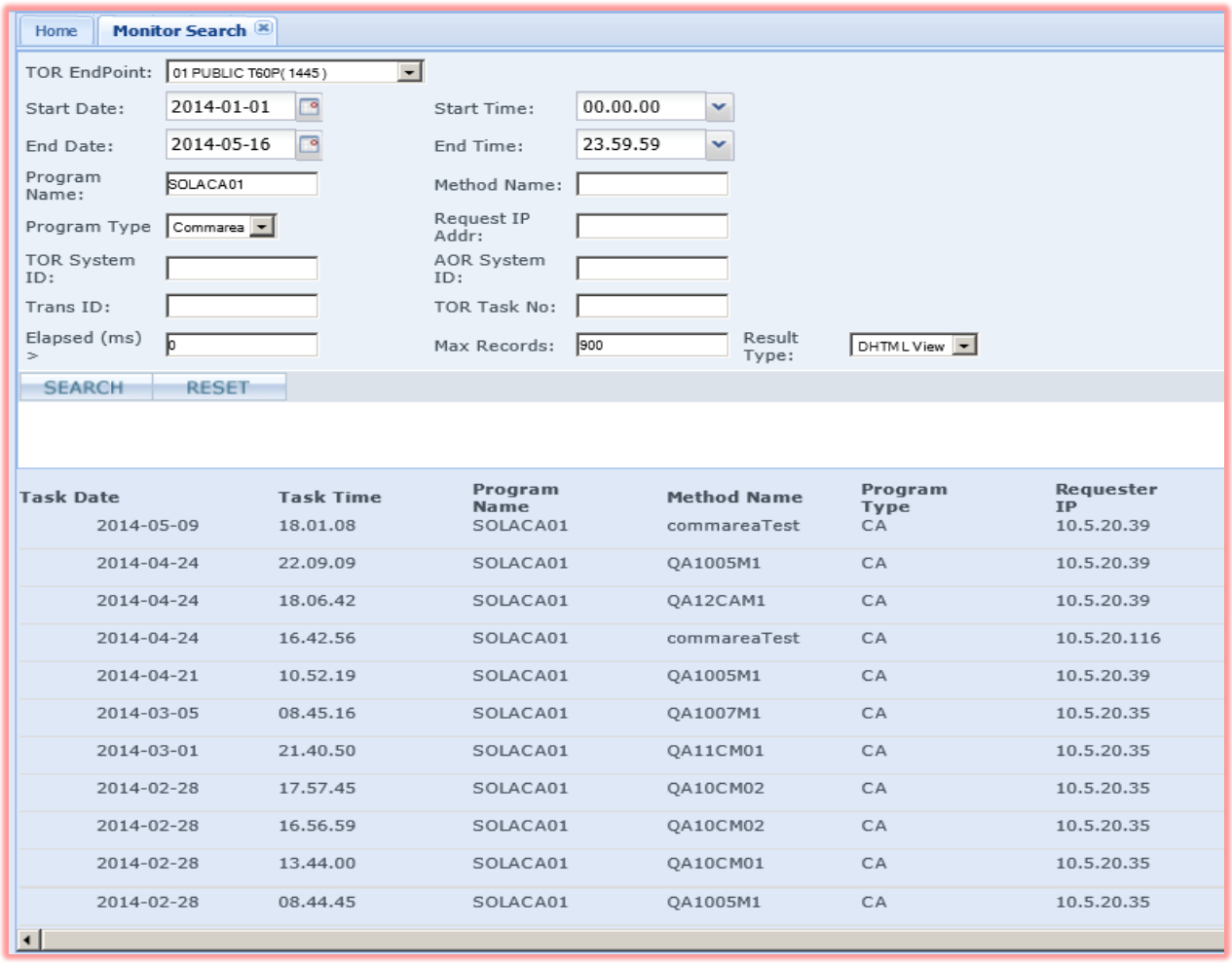
- **Elapsed (ms):** filter's the search to transaction's that are long running based on 'Elapsed (ms)' by specifying the threshold for filtering the records based on task elapsed time.

- **Max Records:** specify Max number of records (up to 9999) that need to be extracted for analyzing monitoring data. Monitoring data can be extracted into an Excel spreadsheet with **Result Type** 'EXCEL view'. This gives the flexibility to the administrator to exploit Excel based filtering, pivoting & graphing tools to mine into the monitoring statistics.

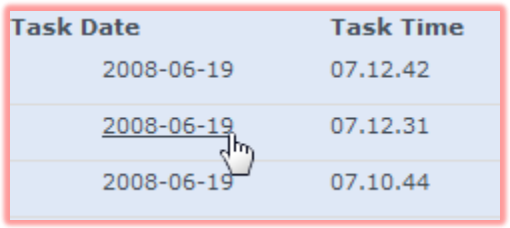
- **Result Type:** specifies how the results will be displayed, either as DHTML (normal view) or as an Excel spreadsheet. Selecting Excel will download the results and open MS Excel (if installed), displaying the data in an Excel spreadsheet.

Once you have specified your search parameters, click SEARCH .

The results of the search will be displayed below the monitor search panel. If the list exceeds the available screen size, then you will need to scroll to see all of the search results.



The information is organized under a series of columns:

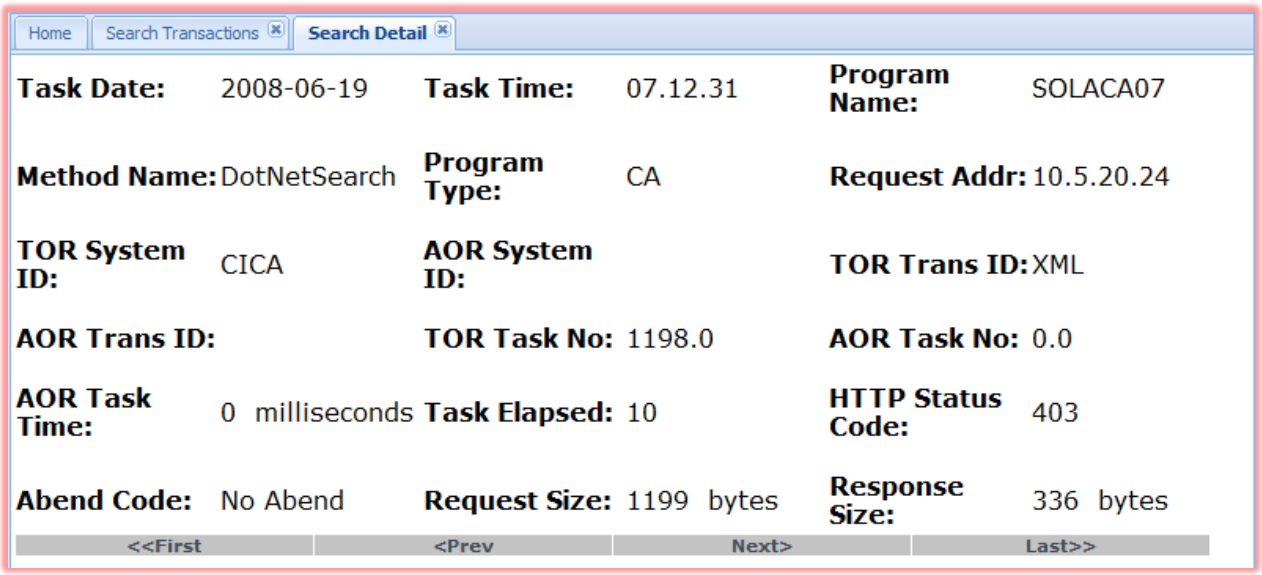


- **Task Date:** the day the transaction was generated, represented as yyyy-mm-dd. Clicking on the date for a specific transaction displays the search details panel that contains very detailed information about the transaction.

- **Task Time:** the time the transaction was generated, represented as hh.mm.ss.

- **Program Name:** the program whose execution generated the transaction.
- **Method Name:** the name of the method whose execution generated the transaction.
- **Program Type:** the category (type) of program whose execution generated the transaction.
- **Requester IP:** the IP Address of the originating request (responsible for executing the method that generated the transaction, if it comes via HTTP).

To get detailed information about a specific transaction, click on the transaction date. This will display the search detail panel.



This panel contains detailed information about a specific transaction organized under the following headings:

- **Task Date:** the date (yyyy-mm-dd) of the transaction.
- **Task Time:** the time (hh.mm.ss) of the transaction.
- **Program Name:** the program whose execution generated the transaction.
- **Method Name:** the method whose execution generated the transaction.
- **Program Type:** the type of the program whose execution generated the transaction.
- **Request Addr:** the IP Address of the originating request (responsible for executing the method that generated the transaction).

- **TOR System ID:** unique identifier for the TOR region where the transaction originated.
- **AOR System ID:** unique identifier for the AOR region where the transaction originated.
- **TOR Trans ID:** unique identifier given to each program that runs in a TOR.
- **AOR Trans ID:** unique identifier given to each program that runs in an AOR.
- **TOR Task No:** unique identifier that is given to each unique instance of a program running in a TOR.
- **AOR Task No:** unique identifier that is given to each unique instance of a program running in an AOR.
- **AOR Task Time:** how long it took to execute the program in the AOR, accurate to +/- 5 milliseconds.
- **Task Elapsed:** the total end to end time (AOR+TOR) that it took to execute the program, accurate to +/- 5 milliseconds.
- **HTTP Status Code:** the HTTP response code generated as a result of the transaction (e.g. 200 – OK, 403 – Auth Failure, etc.)
- **Abend Code:** the mainframe abend code if the program abnormally terminates (i.e. abnormally ends - abends).
- **Request Size:** the size of the input SOAP XML in bytes.
- **Response Size:** the size of the output SOAP XML in bytes.

The links at the bottom of the panel allow you to navigate through all the transactions in the list.

<<First: show details for the first transaction in the list.

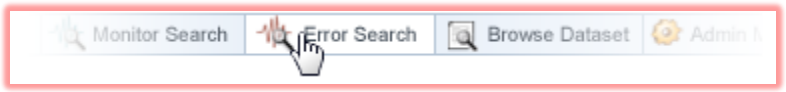
<Prev: show details for the previous transaction.

Next>: show details for the next transaction.

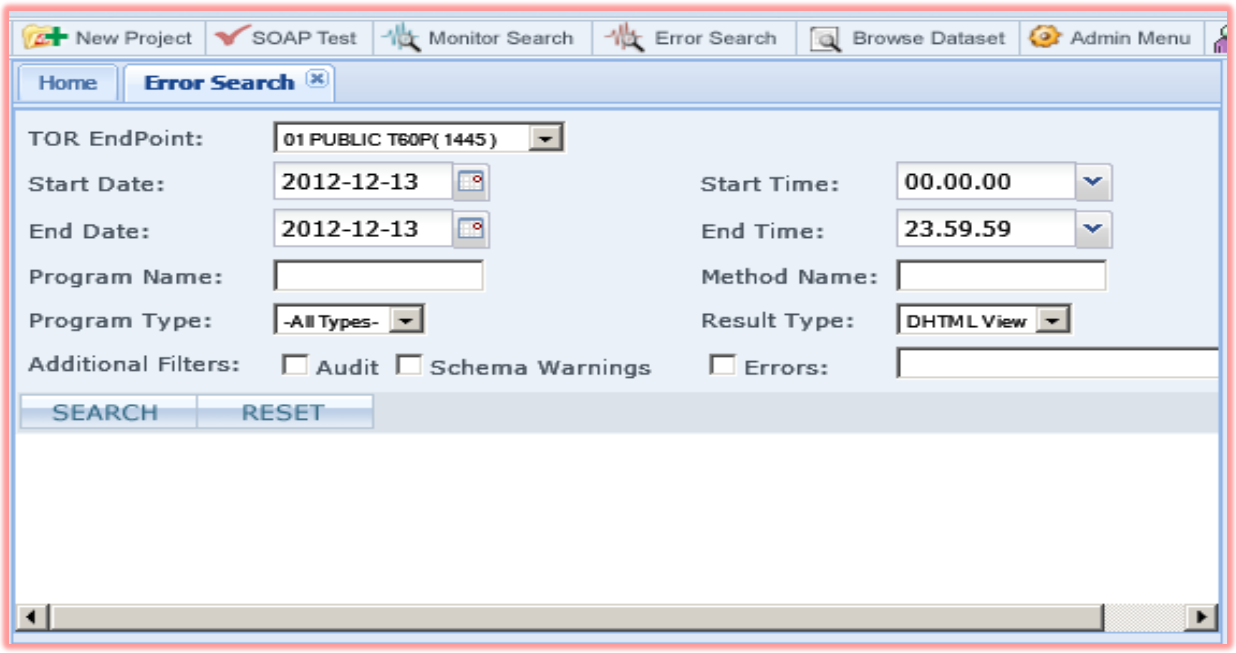
Last>>: show details for the last transaction.

Error Logs

You can search the error logs by clicking the **Error Search** button on the button bar.



This will display the error search panel.



To conduct a search of the error log, enter search parameters using the search fields to narrow the scope of your search. You can also conduct a search with the default (mostly blank) settings.

The following is a description of the search fields:

- **TOR EndPoint:** narrows the search to errors generated within a matching TOR region.
- **Start Date and End Date:** the start and end dates are automatically populated with the current date, though these values can be changed if necessary. All errors are stamped with the date and time at which they take place, and only errors that took place on or after the start date and on or before the end date will be returned.
- **Start Time and End Time:** the start and end times are automatically populated with the current system time and can be changed by manually entering a time (hh.mm.ss). All errors are stamped with the date and time at which they take place, and only errors that took place at or after the start time and at or before the end time will be returned.
- **Program Name:** narrows the search to errors generated by the specified program.
- **Method Name:** narrows the search to errors generated by the specified method.

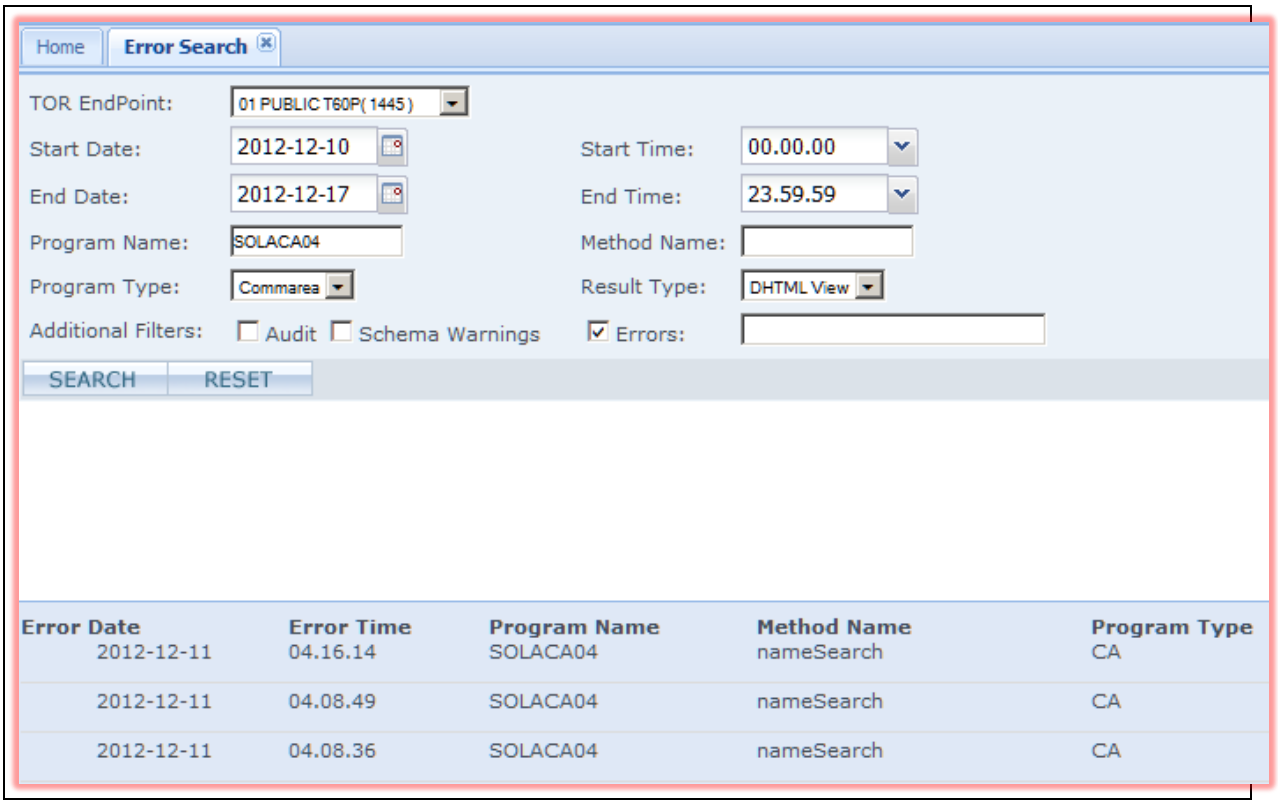
- **Program Type:** narrows the search to errors generated by a method executed by the specified program type. Options are All Types, Commarea, Callable, BMS3270, Outbound, AdhocSQL, TgadpXml or Custom.

- **Result Type:** specifies how the results will be displayed, either as html (normal view) or as an Excel spreadsheet. Selecting Excel will download the results and open MS Excel (if installed), displayed the data in an Excel spreadsheet.

- **Additional Filters:** narrows the search to include only Audit Information, Schema Warnings or specific Error codes.

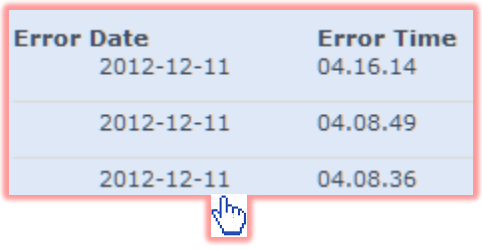
Once you have specified your search parameters, click. SEARCH

The results of the search will be displayed below the error search panel. If the list exceeds the available screen size, then you will need to scroll to see all of the search results.



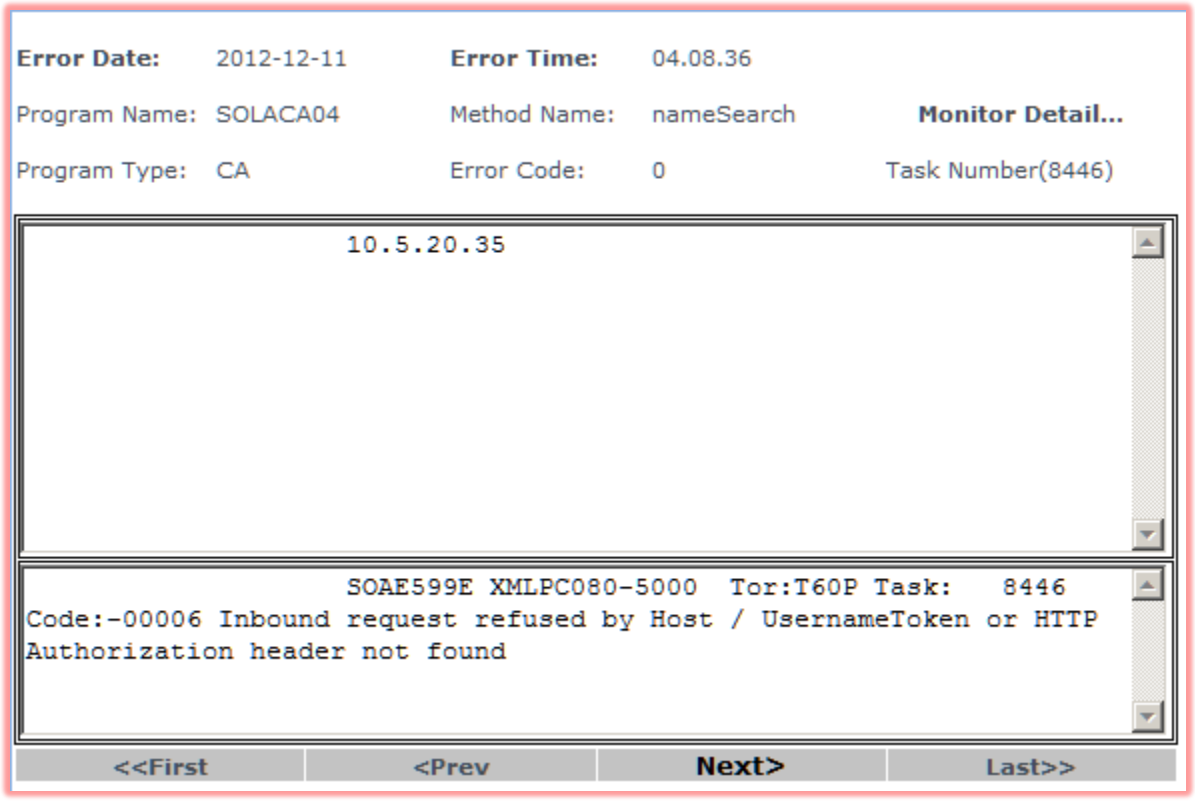
.
The information is organized under a series of columns:

- **Error Date: :** the day the error was generated, represented as yyyy-mm-dd. Clicking on the date for a specific error displays the search details panel that contains very detailed information about the error.



- **Error Time:** the time the error was generated, represented as hh.mm.ss.
- **Program Name:** the program that generated the error.
- **Method Name:** the name of the method that generated the error.
- **Program Type:** the category (type) of program that generated the error.

To get detailed information about a specific error, click on the error date. This will displays the search detail panel.

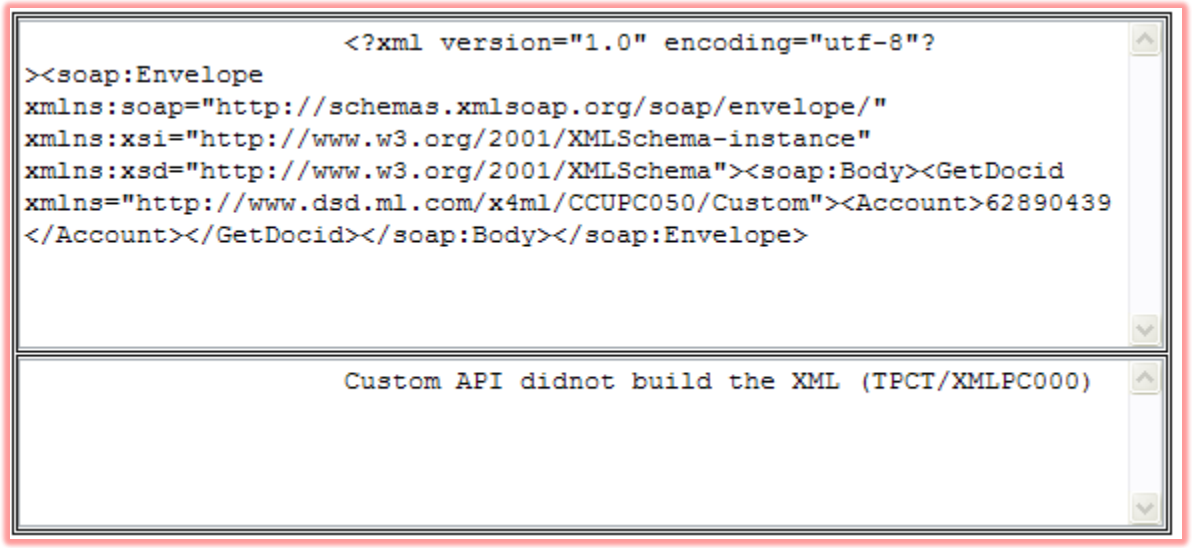


This panel contains detailed information about a specific error organized under the following headings:

- **Error Date:** the date (yyyy-mm-dd) of the error.
- **Error Time:** the time (hh.mm.ss) of the error.
- **Program Name:** the parent program of the method that caused the error.
- **Method Name:** the method that caused the error.

- **Program Type:** the category of the parent program of the method whose execution caused the error.
- **Error Code:** the error code of the generated error.
- **Task Number:** the TOR task number of the task that caused the error.
  the AOR, accurate to +/- 5 milliseconds.

This panel may contain an error display field that contains additional debugging information.



This field is divided into two panes. The bottom pane displays the mainframe error message, while the top pane displays the input XML that caused the error.

The links at the bottom of the panel allow you to navigate through all the errors in the list.

<<First: show details for the first error in the list.

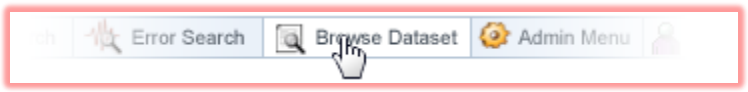
<Prev: show details for the previous error.

Next>: show details for the next error.

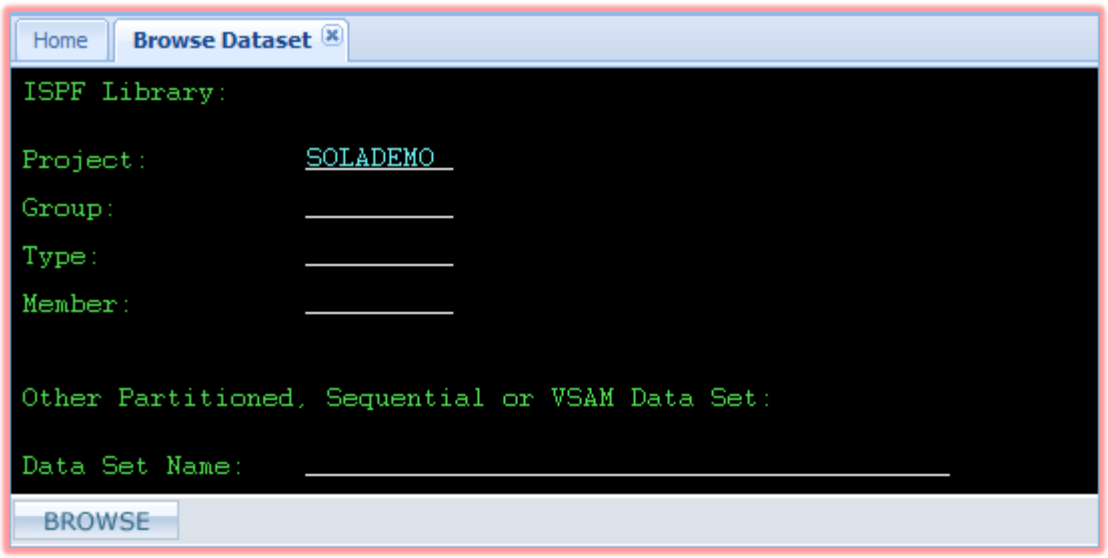
Last>>: show details for the last error.

Dataset Browsing

SOLA Developer has a facility that allows users to browse mainframe datasets. To access this facility, click the **Browse Dataset** button on the button bar.



This will display the browse dataset panel.



This panel is designed to look and function just like a mainframe terminal. The panel is comprised of a series of fields that you can use to enter information about the sequential dataset or PDS member that you wish to brose.

When you have filled out the required fields, click 

.

The following is a description of the panel's fields:

- **Project, Group, Type** and **Member:** these fields are used for searching for PDS members. The majority of mainframe PDS names use three qualifiers, Project, Group and Type. These fields are required when attempting to view PDS members.

- **Fully Qualified Name:** this field is used when searching for sequential datasets of PDSs. This field is required attempting to view sequential datasets.

Orchestration

Orchestration is a separately priced option. Documentation on the Orchestration feature will be supplied on request.

Administration

SOLA Developer is equipped with a comprehensive suite of administrative functions that pertain to the development tool, it's environment variables and access controls.

There are two administration consoles, the admin menu and the access controls menu:

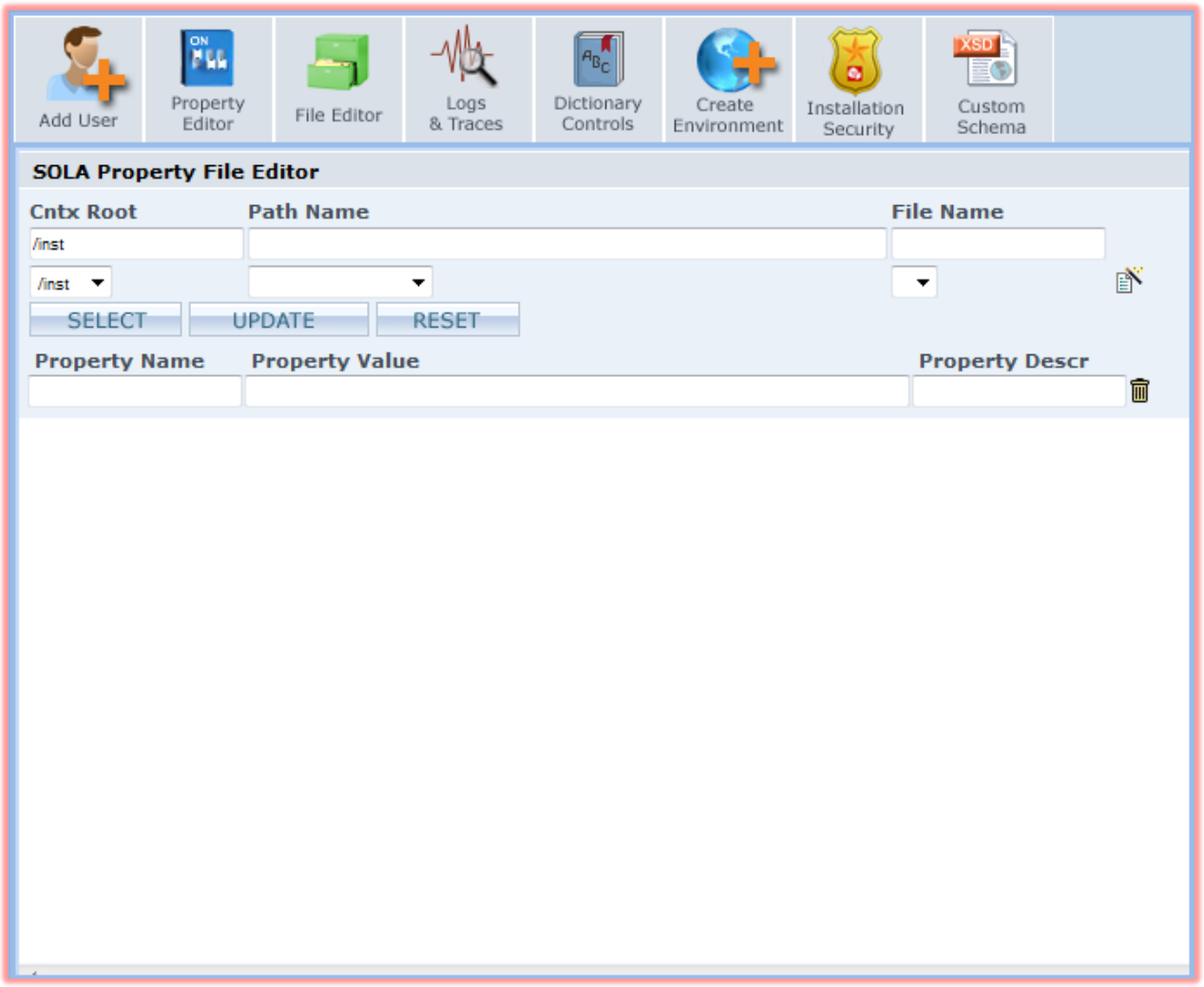
- **Admin Menu:** this console contains administrative tools for configuring system files and properties, managing dictionary settings, viewing log files and creating custom schemas.

- **Access Controls Menu:** this console contains administrative tools for managing access control groups, user access lists and alternate ids, as well as accessing the user activity log.

Admin Menu

To access SOLA Developer's administration functions, click on the **Admin Menu** button in the button bar.

This will display the admin console.



The admin console is comprised of five panels, accessed by icon tabs. The default icon tab is **Property File Editor**. Click on a different icon tab to open the other admin panels.

This icon tab opens user registration panel, where you can register a new SOLA user.

This icon tab opens the file editor panel, which you can use to browse and edit system files such as debugging.xml and endpoints.xml. This is very similar to the property editor panel, except that it displays and allows you to edit the entire file, rather than the properties from that file.

This icon tab opens the property editor panel, which you can use to browse and edit system properties, such as UDDIPassword, FTPSite, SOLASOAPAddress and more. This panel is very similar to the file editor panel, except that it extracts properties from a system file and displays only those properties rather than the entire contents of the file.

This icon tab opens the dictionary control panel where you can make changes to the various global dictionaries, upload dictionary files and download global dictionary files to your local machine.

This icon tab opens the logs and traces panel, where you can gain quick access to SOLA log and trace files.

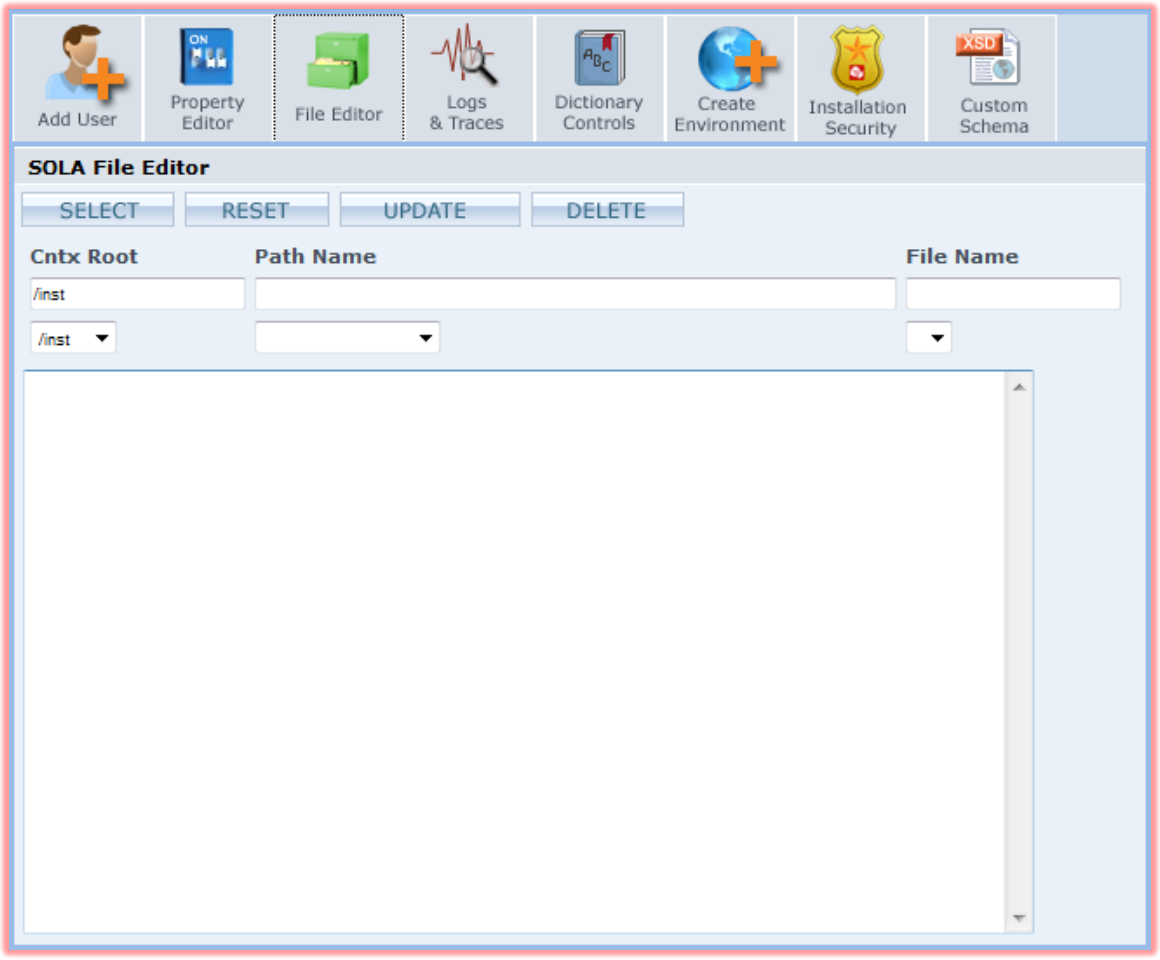
This icon tab opens the Create Environment panel, which you can use to create custom environments (test, production, etc.).

This icon tab opens the custom schema panel, where you can configure new or existing custom properties for projects, users, programs and more.

This icon tab lets you change the SOLA installation password (SOLAIN) when logged in either as SOLAIN or an Administrator.

File Editor

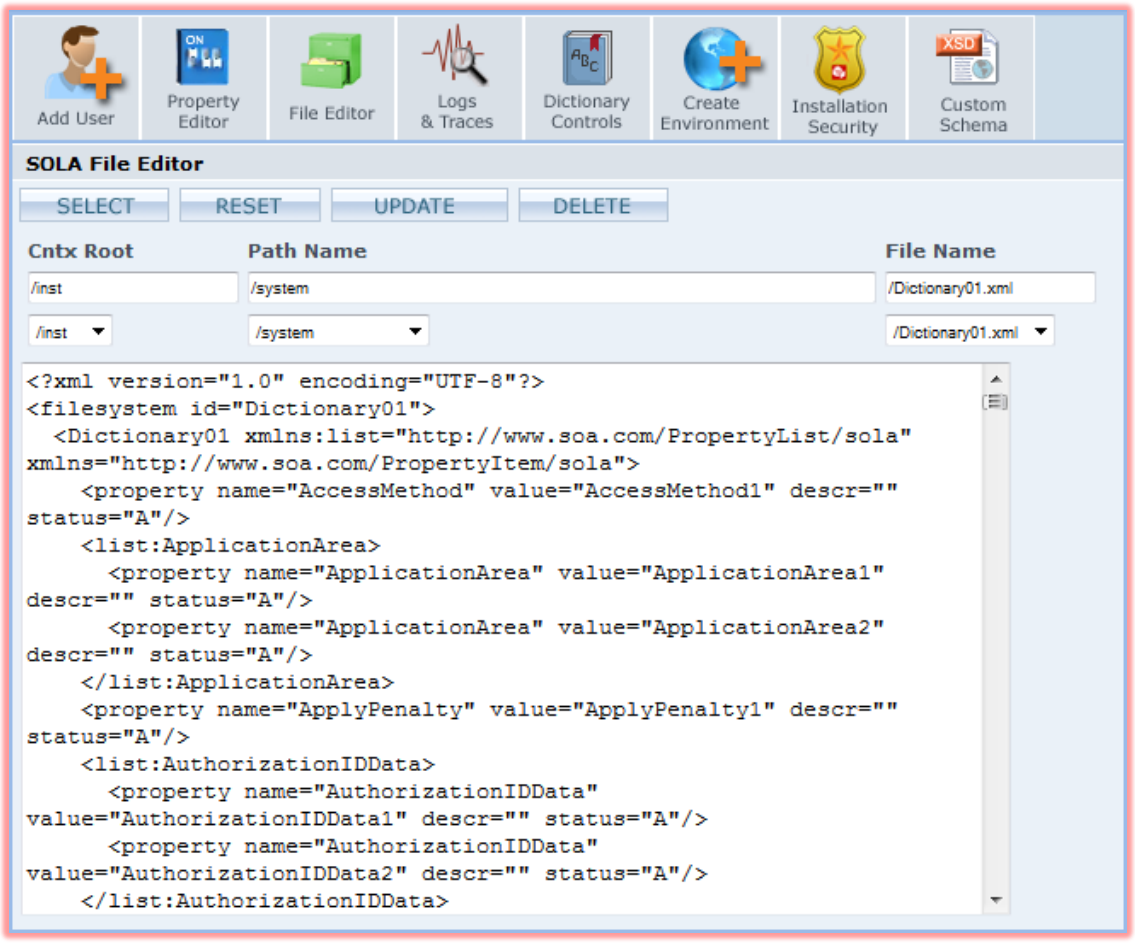
The Property / File Editor panels are very similar to one another and serve the same purpose; the editing of system files. The file editor provides access to the entire file, whereas the property editor extracts system properties from the file you are editing and displays them as fields. Which of the two you use depends on your own preferences.



The top portion of the screen is used to locate the file you wish to view or change. You can either manually enter information into the upper fields, or use the lower menus to select locations from a list. Start with the CntxRoot menu to select a root. Doing so will populate the other two menus.

Once you have located a file, click SELECT .

The contents of the file will be displayed in the large text box.



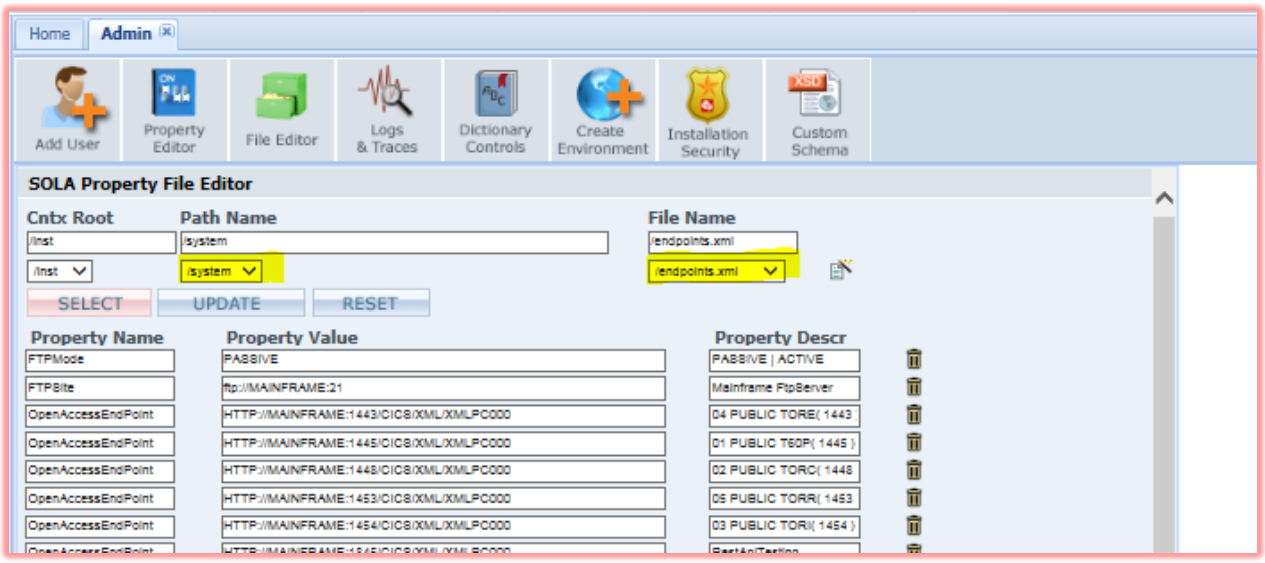
When you wish to save your changes, click UPDATE .

If you want to undo your changes, click RESET .

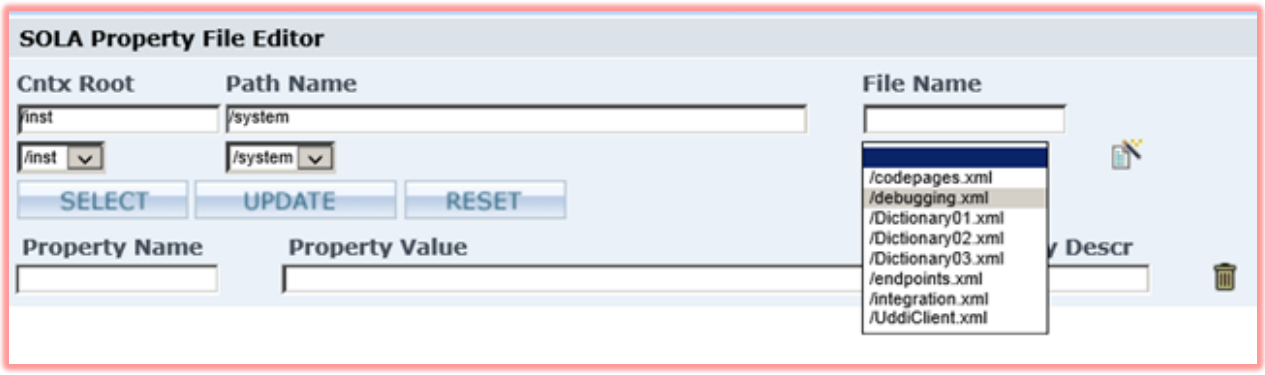
If you want to delete the file, click DELETE . This action is not undoable!

Property Editor

The Property / File Editor panels are very similar to one another and serve the same purpose, the editing of system files. The file editor provides access to the entire file, whereas the property editor extracts system properties from the file you are editing and displays them as fields. Which of the two you use depends on your own preferences.



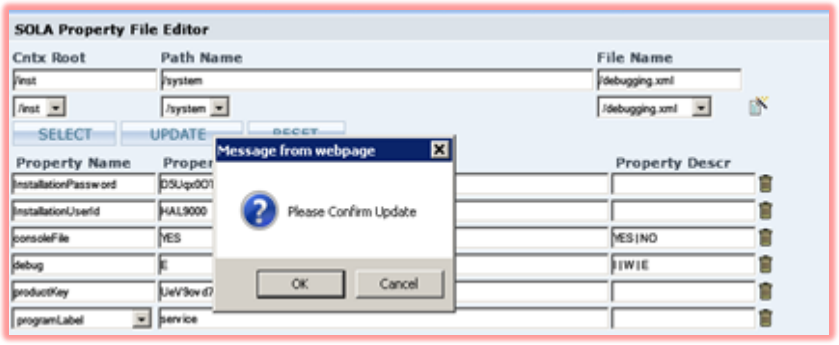
The top portion of the screen is used to locate the file whose properties you wish to view or change. You can either manually enter information into the upper fields, or use the lower menus to select locations from a list. Start with the CntxRoot menu to select a root. Doing so will populate the other two menus.



Once you have located a file, click 

.

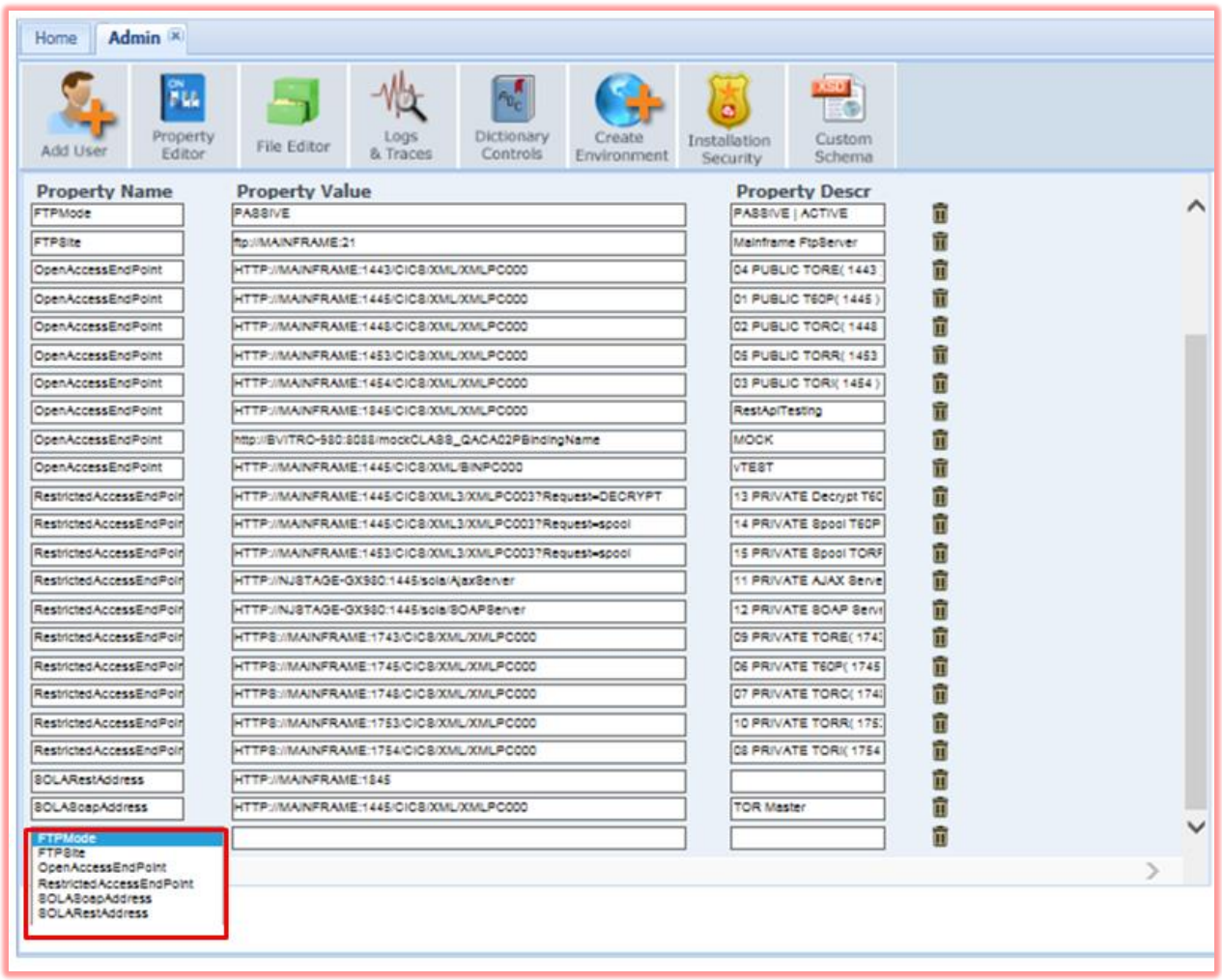
The file's properties will be displayed as fields in the property editor. You can make whatever changes you want by changing the values in the Property Name fields, or adding a Property Value and Description to a blank field.



Note: For a complete description of each Property Name and their allowable values please refer to the SOLA Administration Guide.

To add a blank row so that you can create a new property, click the icon and select a property name from the drop down.

To delete an existing blank row or property, click that row's corresponding icon.



When you wish to save your changes, click 

and then confirm the Update by clicking **OK.**

If you want to undo your changes, click 

.

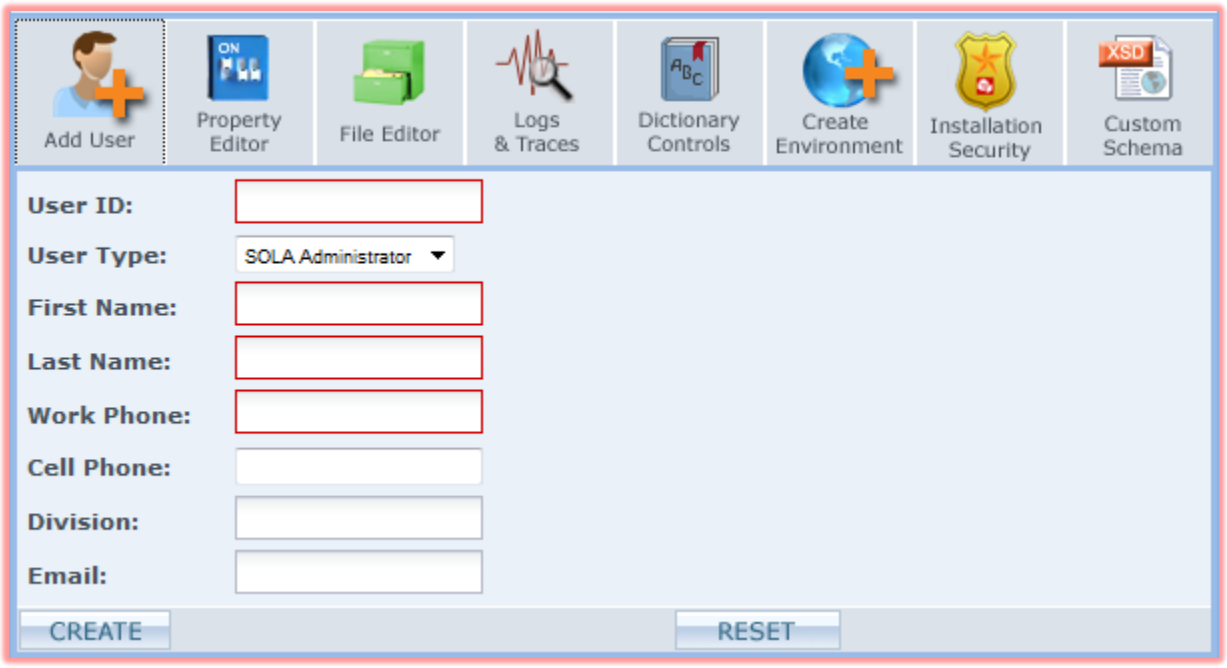
If you want to delete the file whose properties you are editing, click 

.

WARNING: this action is not undoable.

Add User

The Add User panel is used to register a user account with SOLA Developer. Once you successfully log in, you must enter the required registration information to use SOLA Developer.

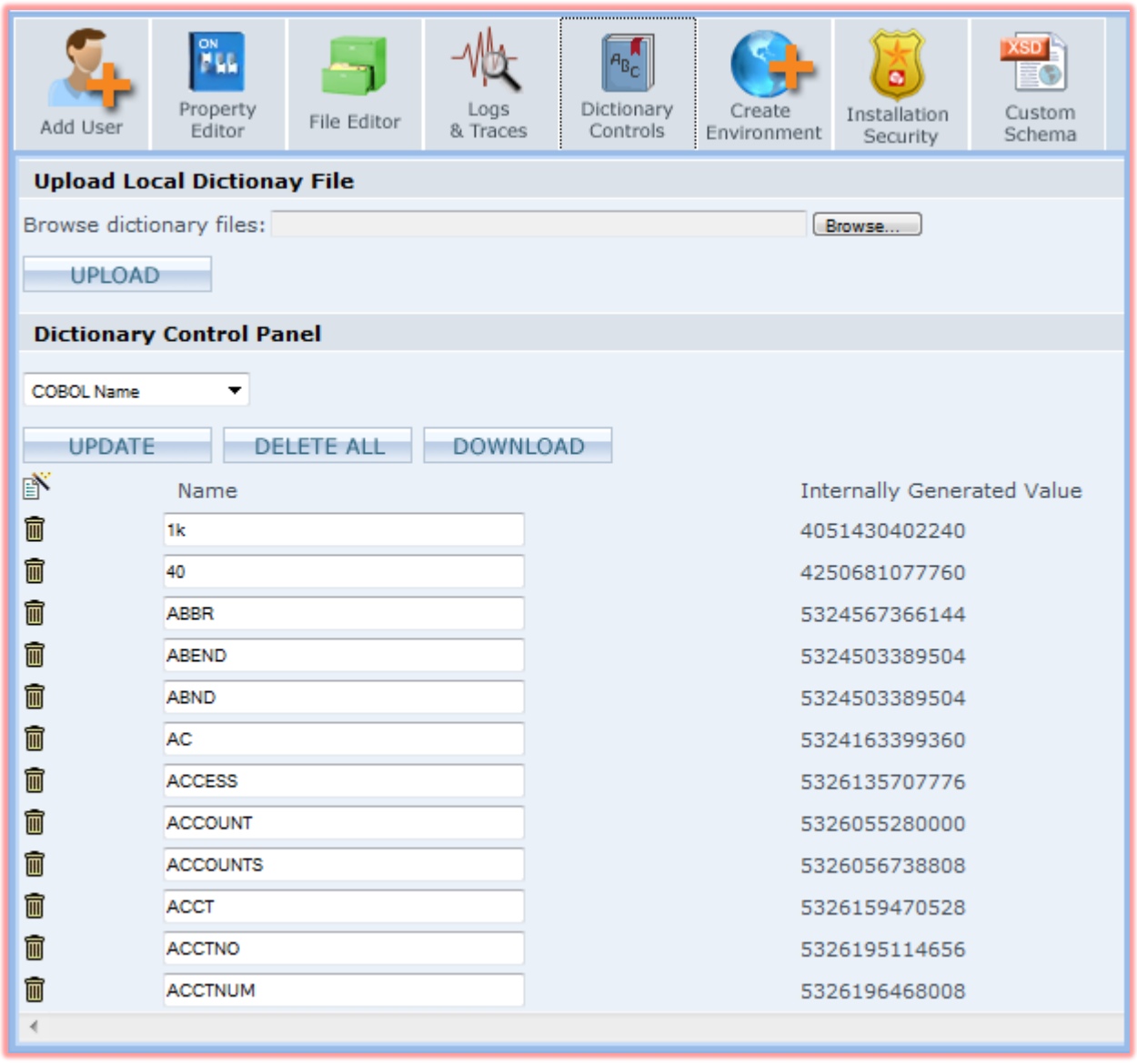


Your first name, last name and work phone number are required. You can also supply your cell phone, division and email address.

Dictionary Controls

The SOLA Dictionary is used during analysis to replace cryptic variable names with names that are more easily understood. For example, a COBOL variable called "LK-CLNT-NM" could be replaced with "ClientName". Once values are defined, SOLA can attempt to automatically replace matching variable names with the specified definitions or to present a drop down menu with close matches if more than one match is found.

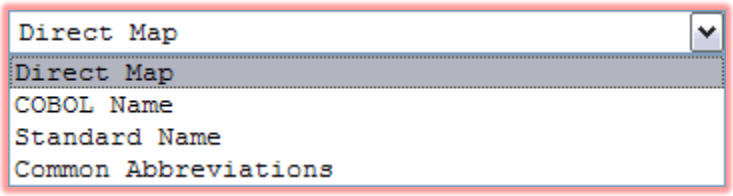
Note: Throughout this section all references to COBOL also apply to PL/I.



The dictionary panel is divided into two sections, the main dictionary control panel below and the upload section above.

Selecting Dictionary Type

The Dictionary menu is used to select the data from one of the four SOLA dictionaries.



- **Direct Map:** shows a list of COBOL field names and the equivalent schema names that they should be substituted for.

- **COBOL Name:** shows an internally maintained list of tokens that are used by the dictionary to conduct match searches. A token is a part of a COBOL field name that is derived by splitting the name at the hyphens (or underscores in the case of PL/I). For example, the COBOL field name WS-ACCT-NUM consists of three tokens, WS, ACCT and NUM. This list contains two columns, Name and Internally Generated Value. The Name column contains the COBOL token, and the Value column contains an internally calculated number that SOLA's heuristic algorithm calculates. Although this is an internal list, it can be added to.

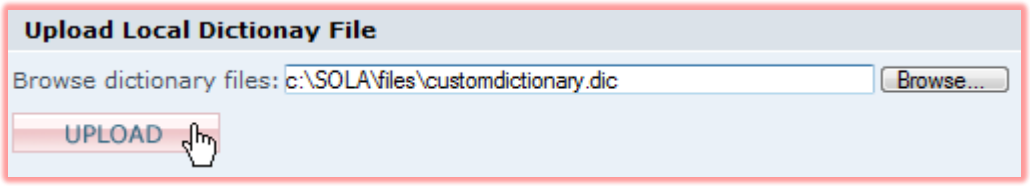
- **Standard Name:** shows an internally maintained list of tokens that are used by the dictionary to conduct match searches. A token is a part of a schema name that is derived by splitting the name at the capitalized letters. For example, the schema name AccountNumber consists of two tokens, Account and Number. This list contains two columns, Name and Internally Generated Value. The Name column contains the standard token, and the Value column contains an internally calculated number that SOLA's heuristic algorithm calculates. Although this is an internal list, it can be added to.

- **Common Abbreviations:** This dictionary contains a list of abbreviations used in SOLA's tokenizing processing logic for parsing COBOL copybook fieldnames. If a COBOL token matches a common abbreviation, then that abbreviation will be substituted for the token.

Uploading and Downloading Dictionary Files

The SOLA dictionary has the capability to add to its contents by uploading text files and to export its contents by allowing users to download the contents as a text file. Uploaded files are applied to the currently selected dictionary type (Direct Map, COBOL Name, etc.) and are appended to existing data. When downloading, only the currently selected dictionary is exported.

The file format for both downloaded files and files to be uploaded is as follows: .txt file, plain ASCII text, ~ (tilde) delimited and carriage return separated.



To upload a dictionary file (and append that file to the existing dictionary), either manually enter a file name (including full path) or use the **Browse** button to locate a file using Windows explorer.
When you are finished, click UPLOAD .

To download the currently selected dictionary, click DOWNLOAD .

Working with the Dictionary Control Panel

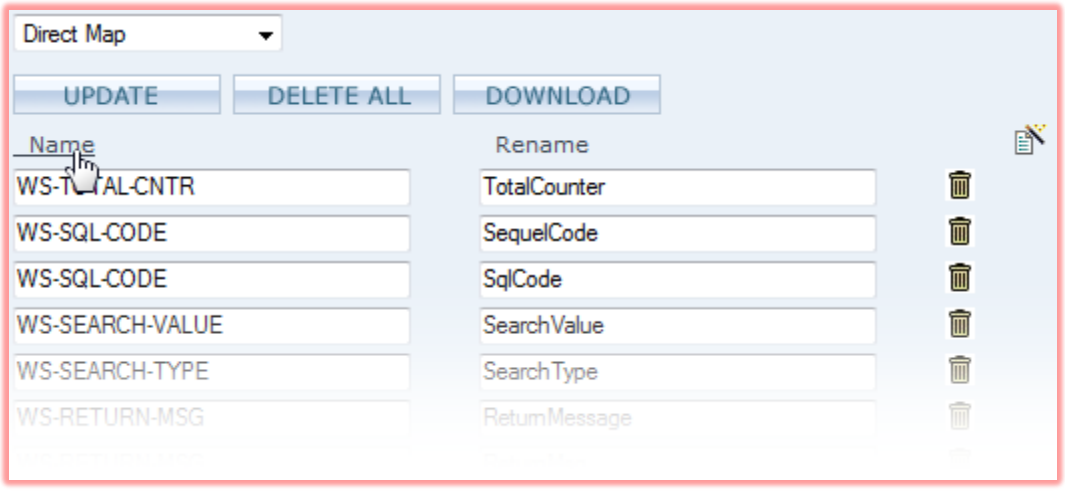
The data shown on the Add to Dictionary screen is organized under two column headings. The column on the left is always called **Name**, while the column on the right can be called either **Rename** or **Internally Generated Value**, depending on which of the four dictionary types you are looking at.

- **Name:** this column contains a list of variable names that need to be replaced with more human-readable names when performing analysis.
- **Rename:** this column contains the schema equivalent to the value in the Name column. These entries can be defined by the user or automatically populated by SOLA.
- **Internally Generated Value:** this column contains a numerical code generated by SOLA to identify the value in the Name column. This is not changeable and is used only by SOLA.

You can create additional rows of data to add new values to the dictionary, or you can delete either blank rows or existing name/value pairs.

To create additional rows, click the button. To remove an unwanted blank row or to delete a name/value pair, click the button.

You can also make changes to dictionary values (except internally generated values) by modifying the contents of the field you want to change.



You can sort the dictionary by either the "Name" or "Rename" columns in the Direct Map and Common Abbreviations dictionaries and by the "Name" column in the COBOL Name and Standard Name dictionaries. To sort by either column, click once on the column name link to sort in ascending order (a-z) and again to sort in descending order (z – a).

Using Wildcards with the Direct Map Dictionary

The Direct Map dictionary is capable of utilizing wild card characters in its matching algorithm. There are two types of wild card characters used by the Direct Map dictionary; % and ^.

%: the % character can be used either before, after or surrounding a dictionary tag. It is similar to the * character in Windows searches. If it appears before a tag, then any item that ends with that tag will be considered a match. If it appears after a tag, then any item that starts with that tag will be considered a match. If it surrounds a tag (appears before and after), then that tag can appear anywhere in the item for it to be considered a match.

Example:

%YZ - Possible matches: XYZ, WXYZ, etc.
WX% - Possible matches : WXYZ, WXY, etc.
%A% - Possible matches: ACCT, BATCH, COMMAREA, etc.

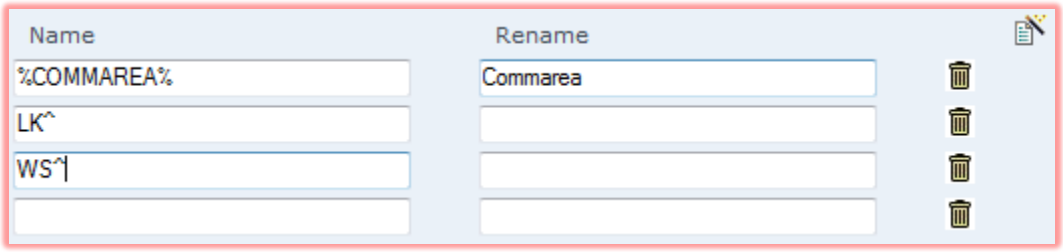
Using the % wildcard character means that every item that matches the tag in the Name column (%XYZ, etc.) will be replaced with the value in the Rename column when the dictionary is applied.

^: the ^ character is used for exclusions at the token level. The dictionary will remove matching dash delimited tokens from COBOL names. When using the ^ character, nothing should be entered in the Rename field (the ^ is for exclusions, not renames).

Example:

LK^ applied to LK-ACCT-NUM would result in ACCT-NUM.

Using the ^ wildcard character means that all tokens matching the tag in the Name column (LK^, etc.) will be stripped from all COBOL names when the dictionary is applied.



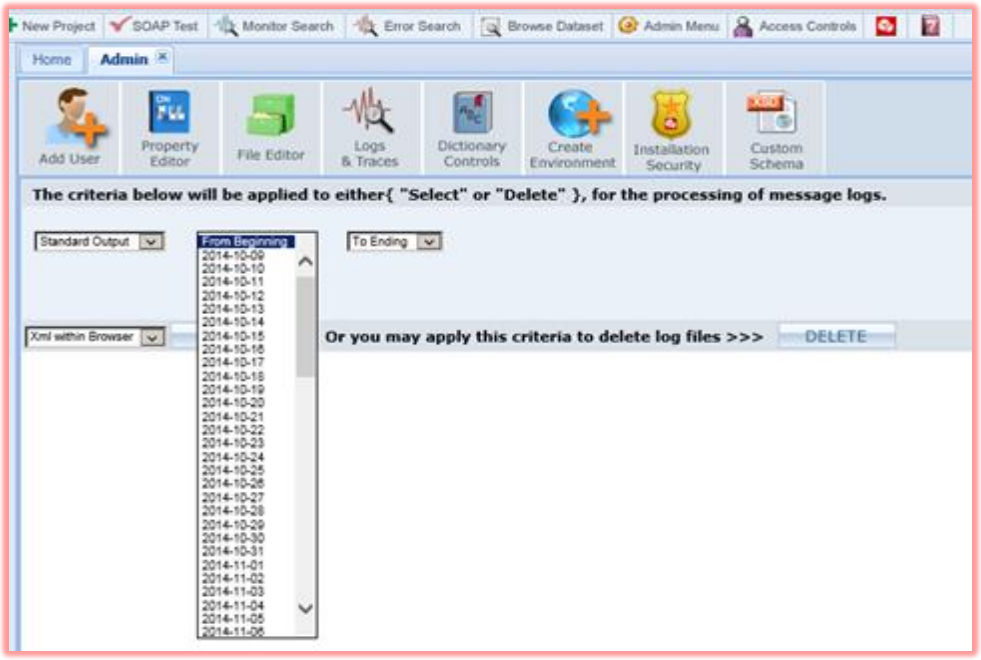
When you wish to save your changes, click UPDATE .

Logs & Traces

The Logs and Trace Files panel provides quick and easy access (view or delete only) to SOLA log and trace files. These same files can be viewed and edited from the Property/File Editor screen. However, if your goal is to view or delete the files as quickly as possible, the Logs and Trace Files screen should be used. These files contain information for SOLA Developer activity only.

The Logs and Trace Files screen contains three pull down menus that are used to quickly select a path and file name.

Once you've selected a file, you can select the date range of the data you would like listed. The default is 90 days if you choose From Beginning.



Select from the dropdown how you want your data to be returned.

View the file by clicking 

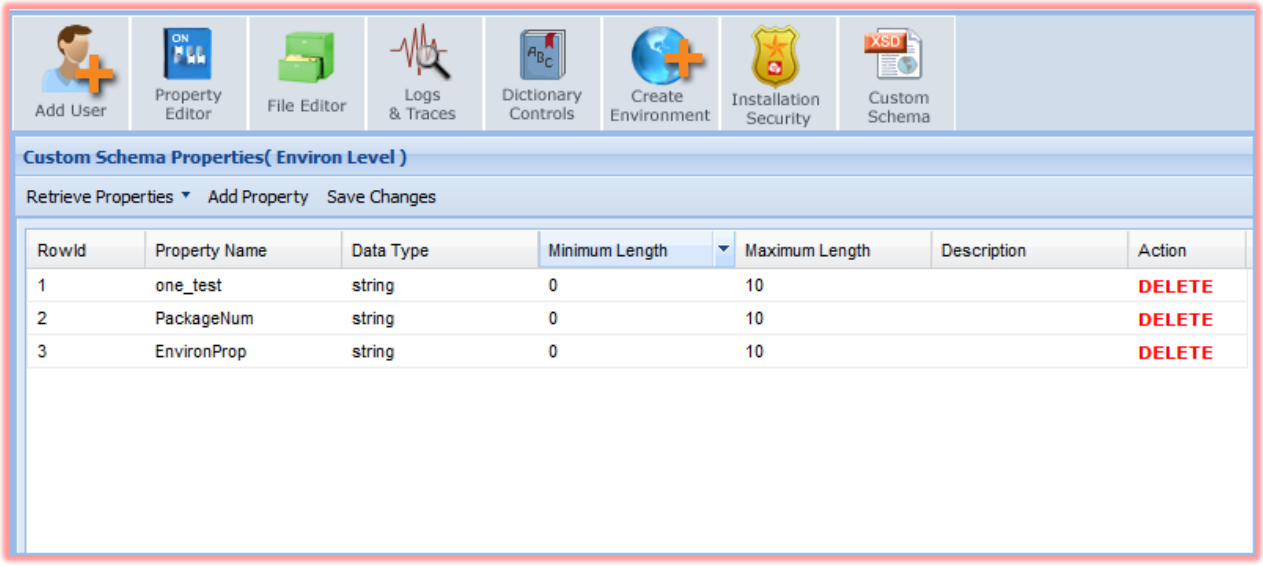
.

You can also delete the file by clicking 

.
WARNING: this action is not undoable.

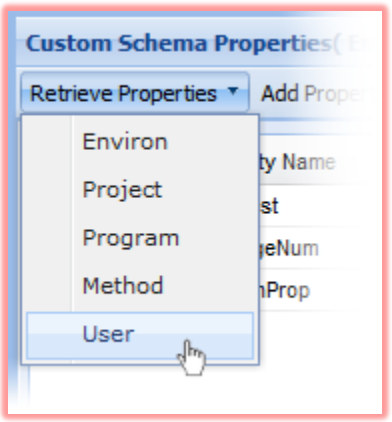
Custom Schema

The custom schema panel allows you to create custom properties for projects, programs, methods, users or environments.



SOLA maintains a schema of properties for each of these categories and the administrator has the opportunity to modify those properties and use the new values. For example, let's say your company uses a home grown change management system called XYZ. In XYZ, all changes are grouped together according to the programmer's hair color. The administrator could define a new user property called "hairColor" and that property could then be used when Finalizing an Analysis to categorize the change for XYZ.

In the image above, the Custom Schema panel is displaying environment properties. To modify user properties, click on the **Retrieve Properties** drop down, and choose **User**.



Options are:

- **Project:** pick this option to create custom properties for projects. These properties will apply to all projects and will be appended to the default properties.

- **Program:** pick this option to create custom properties for programs. These properties will apply to all programs in every project and will be appended to the default properties.

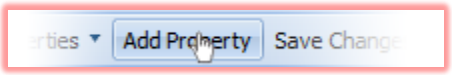
- **Method**: pick this option to create custom properties for methods. These properties will apply to all methods and will be appended to the default properties.

- **User:** pick this option to create custom properties for user accounts. These properties will apply to all users and will be appended to the default properties.

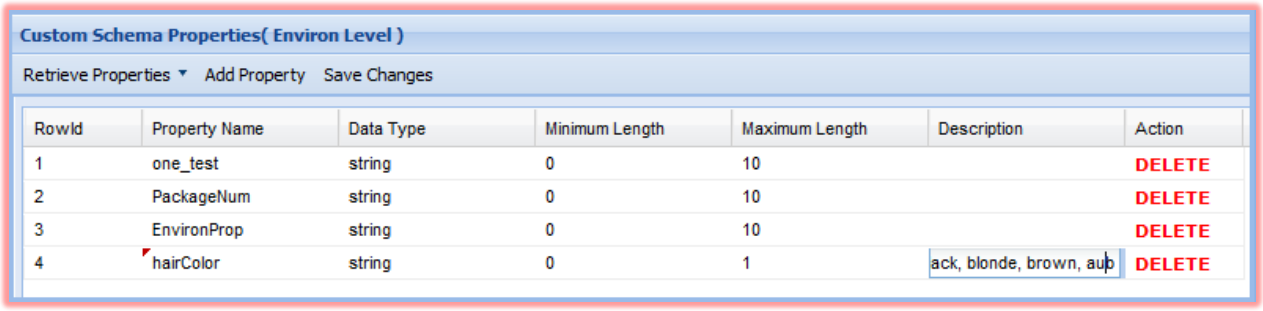
- **Environment:** pick this option to create custom properties for SOLA Developer environments (e.g. Test, Stage, Prod, etc.). These properties will apply to all SOLA Developer environments and will be appended to the default properties.

When you have selected an object to add properties to, the panel will display the existing properties for that object. The properties are organized under columns. These columns correspond to the value fields in the top part of the panel.

- **Rowid:** a sequence number
- **Property Name:** the name of the custom property.
- **Data Type:** the type of data the property can contain. Options are string, int, short and Boolean.
- **Minimum Length:** the property's minimum length in bytes.
- **Maximum Length:** the property's maximum length in bytes.
- **Description:** a free form, optional description.
- **Action:** a DELETE action button.



You can add properties by clicking the **Add Property** button, which adds a blank row at the bottom of the screen. Click inside the blank row and enter values in the value fields. When you have entered all required information, click **Save Changes**. Your new property will be displayed under the property columns.

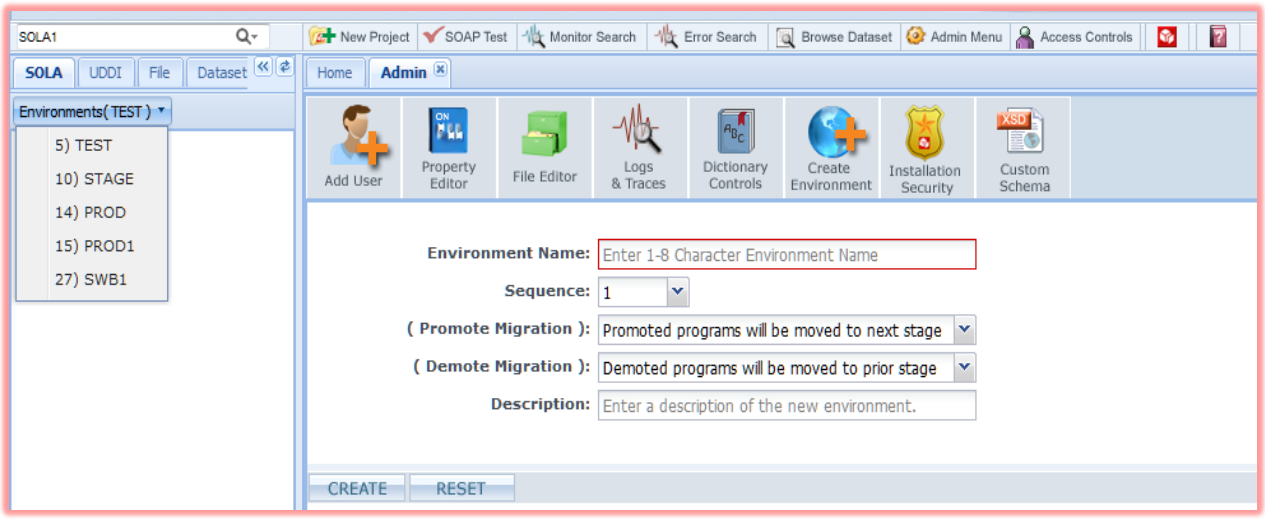


You can delete any property row, including the ones you've just added, by clicking **DELETE** in the Action column.

If you close the panel before clicking the **Save Changes** button, your changes will be lost.

Create Environment

The create environment panel lets you create SOLA environments that are then linked to backend environments using the promote.jcl file.



To create an environment, select an environment code, a sequence, promote and demote option, then enter a brief description such as test, production, etc.

- **Environment Code:** the environment code is a one letter code that represents the environment. The code serves solely as an identifier for the environment, so choose whatever code you want.

- **Sequence:** the sequence represents the environment promotion hierarchy. The lower the sequence number, the lower the environment in the hierarchy. Typically, test environments occupy the lower rungs in the hierarchy, QA or stage environments somewhere in the middle and production environments occupy the highest rungs (and therefore would have the highest sequence numbers). It is recommended that you stagger your sequence numbers (e.g. 1,5 and 9 instead of 1,2 and 3) so that you will have room for additional environments. Sequence numbers do not have to be sequential (1,14, 58 is the same as 1,2,3).

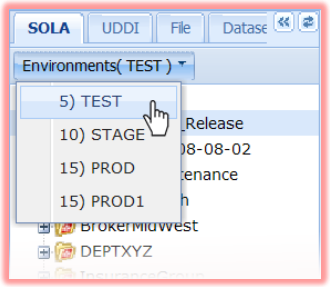
- **Promote Migration:** choose to 'Move' or 'Copy' programs to the 'next' stage.

- **Demote Migration:** choose to 'Move' or 'Copy' programs to the 'prior' stage.

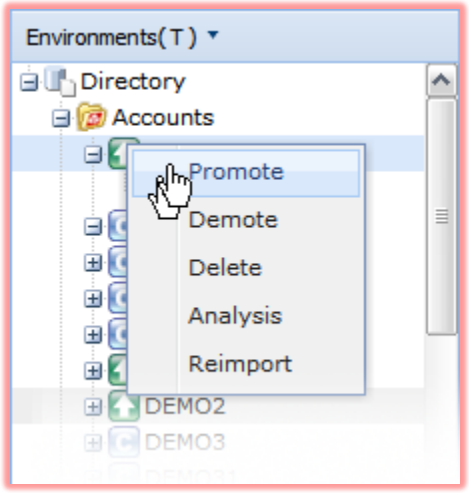
When you have made your selections, click 

to create the environment.

You can filter the view in the SOLA Developer directory to show projects that have programs belonging to a specific environment by using the **Environments** menu.



Once you have created your environments, you need to edit promote.jcl to tie the SOLA Developer environments with the backend environments. The file promote.jcl gets executed whenever you promote or demote a program from one environment to another. For instruction on how to edit promote.jcl, consult the SOLA Administration Guide.



To Promote or Demote programs, use the directory tree menus. Click on the program name and select 'Promote' or 'Demote' from the menus. If the 'actionOnPromote' or 'actionOnDemote' environment attribute was set to 'M' during the creation of the environment, Promote will advance the program to the next 'higher' environment in the sequence, and likewise, Demote will move the program to the previous or 'lower level' environment in the sequence. After promote and successful 'move' the program object in the lower level is expired.

Note: One exception by default is Demote from the highest level (e.g. Prod) will always be forced to 'copy'. After Demote and successful 'move' the program object in the 'higher' level environment is expired, except if it is from the 'highest level' environment (e.g. Prod).

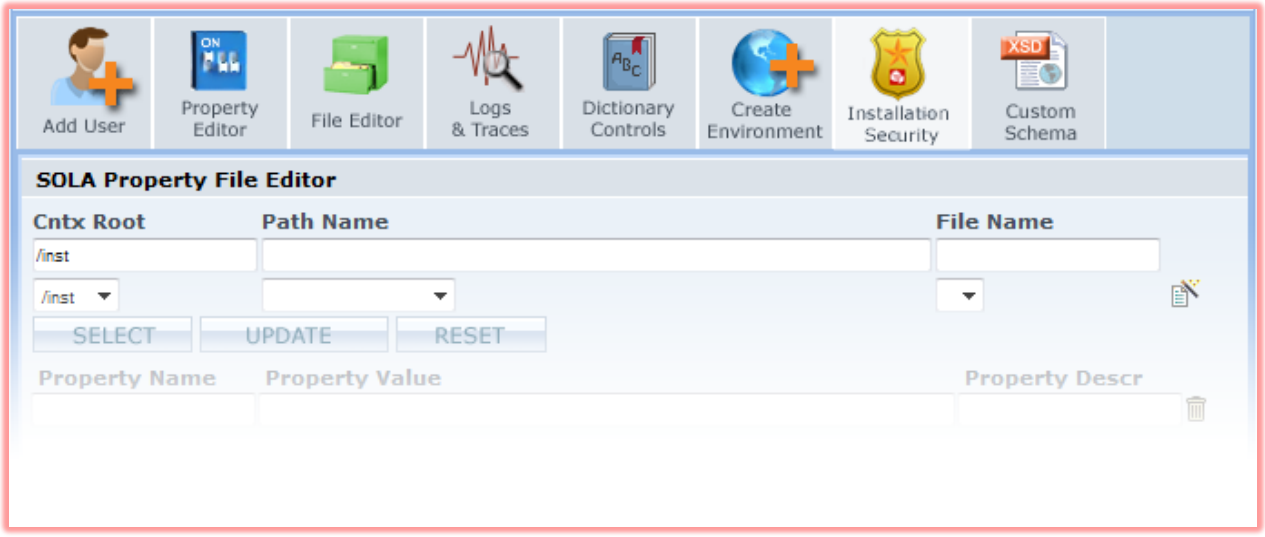
If the 'actionOnPromote' or 'actionOnDemote' environment attribute was set to 'C' during the creation of the environment, Promote will Copy the program to the next 'higher' environment in the sequence, and likewise, Demote will Copy the program to the previous or 'lower level' environment in the sequence. After promote and successful 'copy' the program object in the 'lower level' is retained, and after demote and successful 'copy' the program object in the 'higher level' is retained.

Deleting Environments

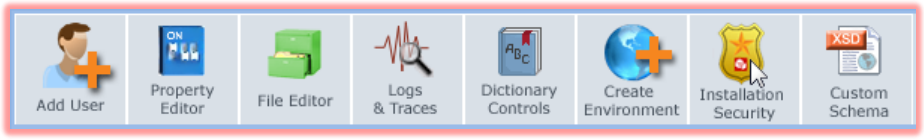
You cannot delete environments using SOLA Developer. You will need to use Resource Manager.

Installation Security

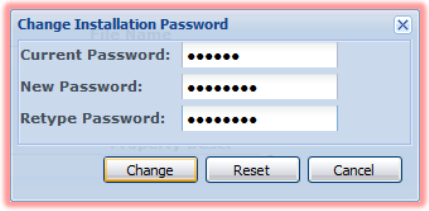
The installation security icon tab lets you change the change the SOLA installation password (SOLAIN) when logged in either as SOLAIN or an Administrator.



To change the installation password, go to the Admin menu screen and select SOLA Installation Security:



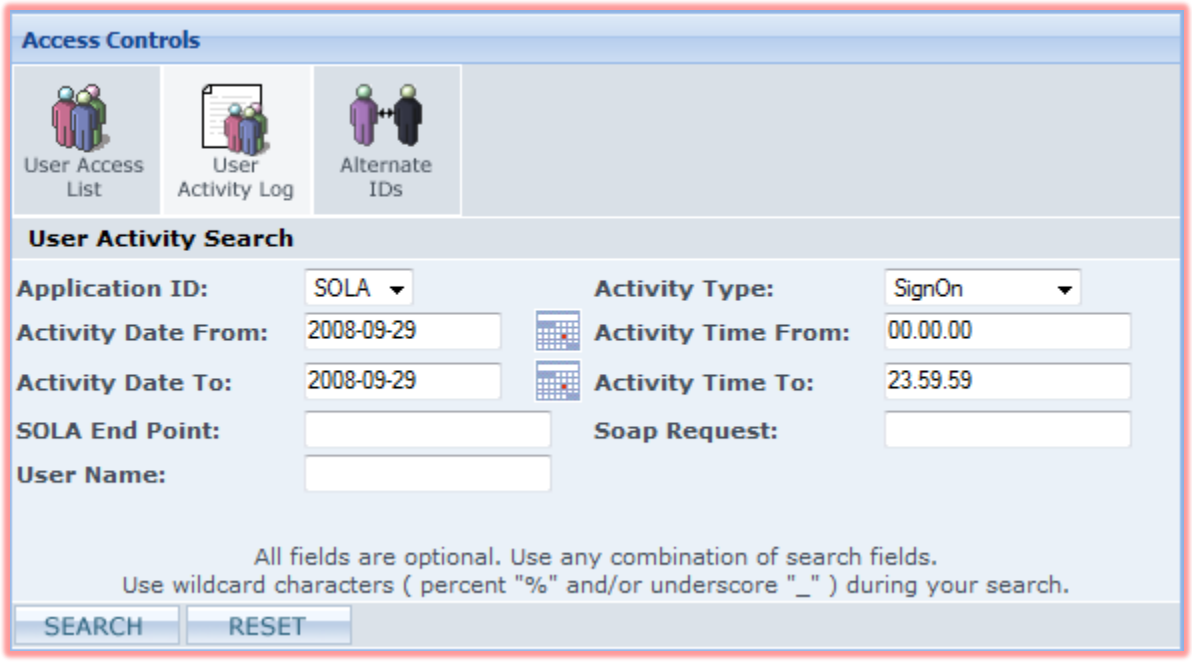
A password change dialog box will appear. Provide the necessary information and click Change.



Access Controls

To access SOLA Developer's access control functions, click on the **Access Controls** button in the button bar.

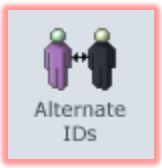
This will display the access controls console.



The access controls console is comprised of four panels, accessed by icon tabs. The default icon tab is **Access Groups**. Click on a different icon tab to open the other access controls panels.

This icon tab opens the user access list panel, which is used in conjunction with the access control groups panel to control user access to regions by selecting which access control groups each user account has access to within each project (i.e. at the project level).

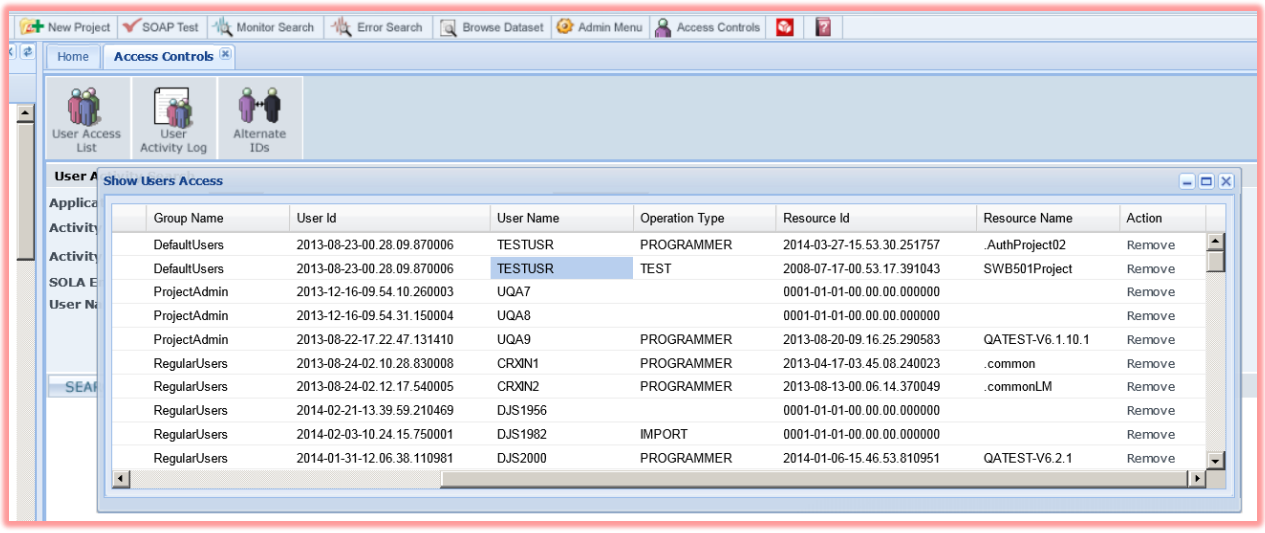
This icon tab opens the user activity search panel which can be used to search the SOLA user activity log for specific activities that match specified search parameters.



This icon tab opens the alternate ids panel, which is used to allow users without mainframe ftp access to make full use of SOLA's functionality (give them ftp access from within SOLA only).

User Access List

The User Access List panel is used to display and/or remove user access.



Each existing SOLA user account is listed in Group Name order, and for each user account, every Project/Resource which the account has access to is listed along with the users Operation Type. SOLA Administrators have global access to all group names (this means all operation types for all projects).

All information displayed on the **Show Users Access** screen is described below.

Note: Columns containing Timestamps are informational only.

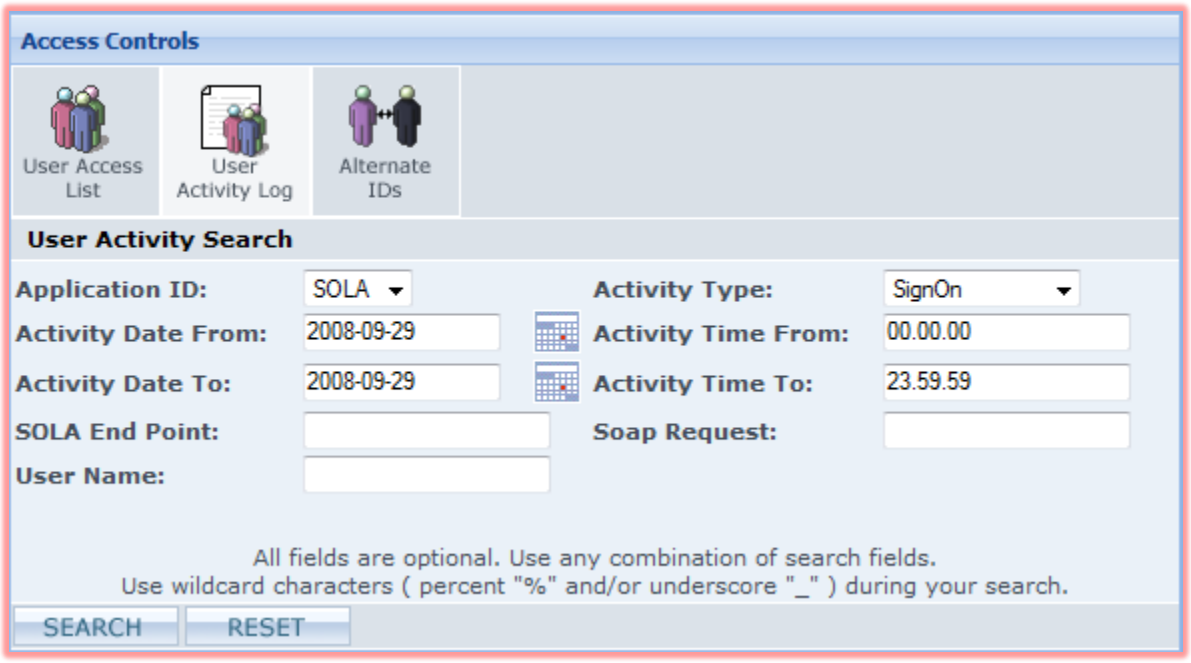
The following describes key User Access information:

- **Group Name:** The User Access Group Name that the User is defined within when the User Access is created.
    - **RegularUsers** – When user access is created in Resource Manager it can be created as a RegularUser or a SOLAAdmin. RegularUser's can later be drag/dropped to another group such as ProjectAdmin's, etc.
    - **DefaultUsers** – A user access is also created in Resource Manager by right clicking at the Directory level and will be placed in the same location as a RegularUser. These type users are listed at the bottom of the User Access List when you click on the Users tab in Resource Manager.
    - **SOLAAdmin** – a user defined here has access to all Groups.
- **User Name:** the name the user was given when it was created and is associated with the user Id.
- **Operation Type:** for each user Id and listed individually on a separate row, the type of operation a user is authorized to perform within SOLA.

- **Resource Name:** Within the Developer Directory tree the project/resource name that each User has access to, to perform certain operations on programs/methods within the project. This access can be defined differently for a user from one project to another. An example of this is a user can be defined as a Programmer in one project with access as a ProjectAdmin to another project.User
- **Action:** The only allowable action here is to remove a user's access from a specific Group or Project by clicking on the **Remove** option.

User Activity Log

The user activity log panel is used to search through SOLA's user activity log in the same way that the monitor search panel (page 226) is used to search SOLA's transaction log.



To conduct a search of the activity log, enter search parameters using the search fields to narrow the scope of your search. You can also conduct a search with the default (mostly blank) settings, though this may take some time to complete and may result in a very long list of activities.

The following is a description of the search fields:

- **Application ID:** currently, the only option is SOLA.
- **Activity Type:** narrows the search to activities of the specified type. Options are:
  - **SignOn:** user sign-on.
  - **Error Search:** error log search.
  - **Monitor Search:** monitor (transaction) search.
  - **Testing:** quick test or raw test.
  - **Import:** importing a program.
  - **Analysis:** method analysis.
  - **Delete:** deletion of a project, program or method.
- **Activity Date From and Activity Date To:** the start (activity date from) and end (activity date to) dates are automatically populated with the current date and can be changed by manually entering a date (yyyy-mm-dd). All activities are stamped with the

date and time at which they take place, and only activities that took place on or after the start date and on or before the end date will be returned.

- **Activity Time From and Activity Time To:** the start and end times are automatically populated with the current system time and can be changed by manually entering a time (hh.mm.ss). All activities are stamped with the date and time at which they take place, and only activities that took place at or after the start time and at or before the end time will be returned.

- **SOLA End Point:** narrows the search to activities that involve a request with the specified end point.

- **SOAP Request:** if an activity involves a SOAP request sent through the SOLA website, then this field can be used to narrow the search based on a part of that SOAP request. For example, if you populate this field with the word "SOLA", then any activity that involved a SOAP request with the word SOLA in any context will be returned (provided it matches any other search parameters that are specified).

- **User Name:** narrows the search to the activities of the specified user.

Once you have specified your search parameters, click 

.

The results of the search will be displayed below the activity search panel. If the list exceeds the available screen size, then you will need to scroll to see all of the search results.

The information is organized under a series of columns:

- **Appl ID:** the application involved in the activity. Clicking on the application ID for a specific activity displays the search details panel that contains very detailed information about the activity.

- **Activity Type:** the type of activity. Option are:

  - **SGN:** user sign-on.
  - **MON:** monitor (transaction) search.
  - **LOG:** error log search.
  - **TST:** quick test or raw test.
  - **DEL:** deletion of a project, program or method.
  - **TRC:** a trace initiated by an administrator.

- **Row Number:** each activity is assigned a sequence number, which is displayed here.

- **User Name:** the user involved in the activity.

- **User Date:** the day the activity took place expressed as yyyy-mm-dd.

- **User Time:** the time the activity took place expressed as hh.mm.ss.
- **End Point:** the end point in which the activity took place.

To get detailed information about a specific activity, click on the activity's application ID. This will displays the search details panel.

The search details is a series of fields that contain data about a specific user activity, along with a large field that contains the soap request that was sent to the SOLA soap server as a part of the activity (if there was one).

The information is organized under the following headings:

- **Application ID:** currently, the only option is SOLA.
    - **Activity Type:** narrows the search to activities of the specified type. Options are:
    - **SGN:** user sign-on.
    - **MON:** monitor (transaction) search.
    - **LOG:** error log search.
    - **TST:** quick test or raw test.
    - **TRC:** a trace initiated by an administrator.
- **User Name:** the user account that triggered the activity.
- **Activity Date:** the date (yyyy-mm-dd) of the activity.
- **Activity Time:** the time (hh.mm.ss) of the activity.
- **SOLA End Point:** the mainframe end point where the activity took place.

The links at the bottom of the panel allow you to navigate through all the activities in the list.

<<First: show details for the first activity in the list.

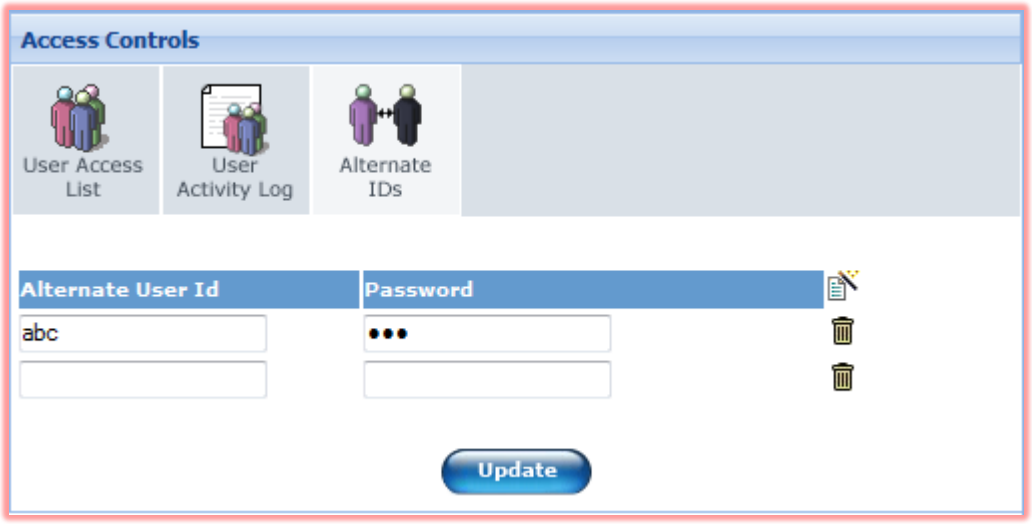
<Prev: show details for the previous activity.

Next>: show details for the next activity.

Last>>: show details for the last activity.

Alternate IDs

The Alternate User IDs panel is used to define mainframe Ids that have FTP access.



Once defined, these IDs can be added to a user Id at the project level to allow users without mainframe ftp access to make full use of SOLA Developer's functionality (give them ftp access from within SOLA Developer only). FTP access is necessary for browsing datasets, importing programs, and for finalizing an analysis.

Fill in the required fields to add an alternate ID.

- **Alternate User Id:** enter a mainframe user Id that has mainframe ftp access. This can, but does not have to, be an existing SOLA user.
- **Password:** the password associated with the ID.

To add additional fields, click the button. To remove an unwanted blank field or to delete an user ID/password pair, click the button.

Appendices

Appendix A: Schema and Copybook Generation

Datatype Mapping and Copybook Generation Rules

The following table lists schema datatypes and how they are handled by SOLA when importing WSDL and generating a copybook to be used for outbound requests.

Datatype	Action	Programmer Response	Notes
string	Gets generated as PIC X(01), unless a maxLength facet is specified in the schema restriction.	User can modify the "len" property during analysis*.	
boolean	Gets generated as PIC X(05). A comment is added to the copybook specifying two possible values.	User can put 'true' or 'false' or '1' and '0'	SOLA Analyzer assumes canonical values. The SOLA runtime doesn't validate this field.
decimal	Gets generated as PIC S9(1) COMP-3, unless totalDigits and/or fractionDigits facets are specified in the schema restriction.	A user can influence this field by changing the "precision" and "scale" properties during analysis.	SOLA validates this field while converting from XML to packed decimal format. Non numeric data is rejected.
float	Gets generated as PIC S9(1) COMP-3, unless totalDigits facet is specified in the schema restriction.	A user can influence this field by changing the "precision" property during analysis.	SOLA validates this field while converting from XML to packed decimal format. Non numeric data is rejected.
double	Gets generated as S9(13)v9(4) COMP-3, unless total Digits facet is specified in the schema restriction.	A user can influence this field by changing the "precision" property during analysis.	SOLA validates this field while converting from XML to packed decimal format. Non numeric data is rejected.

duration	Gets generated as PIC X(256).	A user can influence this field by changing the "len" property during analysis*. We recommend that the user follows the lexical or canonical representation.	This field is treated as a string by SOLA. Validation is not performed.
dateTime	Gets generated as PIC X(25). A comment is added to the copybook showing a sample format such as 2009-11-18T09:27:01.231Z.	A user can influence this field by changing the "len" property during analysis*. We recommend that the user follows the lexical or canonical representation.	This field is treated as a string by SOLA. dateTime validation is not performed.
time	Gets generated as PIC X(14). A comment is added to the copybook showing a sample format such as 12:00:00-05:00	A user can influence this field by changing the "len" property during analysis*. We recommend that the user follows the lexical or canonical representation.	This field is treated as a string by SOLA. time validation is not performed.
date	Gets generated as PIC X(16). A comment is added to the copybook showing a sample format such as 2002-10-10+05:00	A user can influence this field by changing the "len" property during analysis*. We recommend that the user follows the lexical or canonical representation.	This field is treated as a string by SOLA. date validation is not performed.
gMonthDay	Gets generated as PIC X(13). A comment is added to the copybook showing a sample format such as --11-01-04:00	A user can influence this field by changing the "len" property during analysis*. We recommend that the user follows the lexical or canonical representation.	This field is treated as a string by SOLA. month day validation is not performed.
gMonth	Gets generated as PIC X(13). A comment is added to the copybook showing a sample format such as --11-01-04:00	A user can influence this field by changing the "len" property during analysis*. We recommend that the user follows the lexical or canonical representation.	This field is treated as a string by SOLA. month validation is not performed.

gDay	Gets generated as PIC X(5). A comment is added to the copybook showing a sample format such as ---31	A user can influence this field by changing the "len" property during analysis*. We recommend that the user follows the lexical or canonical representation.	This field is treated as a string by SOLA. day validation is not performed.
gYear	Gets generated as PIC X(10). A comment is added to the copybook showing a sample format such as 2009-05:00	A user can influence this field by changing the "len" property during analysis*. We recommend that the user follows the lexical or canonical representation.	This field is treated as a string by SOLA. year validation is not performed.
gYearMonth	Gets generated as PIC X(13). A comment is added to the copybook showing a sample format such as 2009-10+05:00	A user can influence this field by changing the "len" property during analysis*. We recommend that the user follows the lexical or canonical representation.	This field is treated as a string by SOLA. year month validation is not performed.
hexBinary	Gets generated as PIC X(256).	User will have to convert this field programmatically.	This field is treated as a string by SOLA. SOLA doesn't directly support this datatype. No validation is performed
base64Binary	Gets generated as PIC X(256).	A user can influence this field by changing the "len" property during analysis*. Adjust "len" according to the rule that 4 bytes of XML data will get converted to three bytes of binary data, and vice-versa.	SOLA converts Base64 to binary and vice-versa. Base64 validation is performed and an error is thrown if there is a violation.
anyURI	Gets generated as PIC X(256).	A user can influence this field by changing the "len" property during analysis*.	This field is treated as a string by SOLA. No validation is performed
QName	Gets generated as PIC X(256).	A user can influence this field by changing the "len" property during analysis*.	This field is treated as a string by SOLA. No validation is performed

NOTATION	Gets generated as PIC X(256).	A user can influence this field by changing the "len" property during analysis*.	This field is treated as a string by SOLA. No validation is performed
normalizedString	Gets generated as PIC X(256).	A user can influence this field by changing the "len" property during analysis*.	This field is treated as a string by SOLA. No validation is performed
token	Gets generated as PIC X(256).	A user can influence this field by changing the "len" property during analysis*.	This field is treated as a string by SOLA. No validation is performed.
integer	Gets generated as PIC S9(09) COMP.		Numeric validation is performed when mapping from XML to fullword binary.
NonPositiveInteger	Gets generated as PIC X(256).	User will have to convert this field programmatically.	This field is treated as a string by SOLA. SOLA doesn't directly support this datatype. No validation is performed
NegativeInteger	Gets generated as PIC X(256).	User will have to convert this field programmatically.	This field is treated as a string by SOLA. SOLA doesn't directly support this datatype. No validation is performed
nonNegativeInteger	Gets generated as PIC X(256).	User will have to convert this field programmatically.	This field is treated as a string by SOLA. SOLA doesn't directly support this datatype. No validation is performed
long	Gets generated as PIC S9(18) COMP-3.		Numeric validation is performed when mapping from XML to packed decimal.
short	Gets generated as PIC S9(04) COMP.		Numeric validation is performed when mapping from XML to halfword binary.
int	Gets generated as PIC S9(09) COMP.		Numeric validation is performed when mapping from XML to fullword binary.

byte	Gets generated as PIC X(256).	User will have to convert this field programmatically.	This field is treated as a string by SOLA. SOLA doesn't directly support this datatype. No validation is performed
UnsignedByte	Gets generated as PIC X(256).	User will have to convert this field programmatically.	This field is treated as a string by SOLA. SOLA doesn't directly support this datatype. No validation is performed
UnsignedInt	Gets generated as PIC X(256).	User will have to convert this field programmatically.	This field is treated as a string by SOLA. SOLA doesn't directly support this datatype. No validation is performed
UnsignedLong	Gets generated as PIC X(256).	User will have to convert this field programmatically.	This field is treated as a string by SOLA. SOLA doesn't directly support this datatype. No validation is performed
UnsignedShort	Gets generated as PIC X(256).	User will have to convert this field programmatically.	This field is treated as a string by SOLA. SOLA doesn't directly support this datatype. No validation is performed
PositiveInteger	Gets generated as PIC X(256).	User will have to convert this field programmatically.	This field is treated as a string by SOLA. SOLA doesn't directly support this datatype. No validation is performed

Other Restrictions—Numeric Facets

maxLength	is used to define the length of a field during copybook generation
length	is used to define the length of a field during copybook generation
pattern	if the pattern is resolvable length will be derived from it.
enumeration	gets converted to 88 level condition-name in the copybook. If the field type is string the field length will be derived from the maximum length of the enumeration values,
whiteSpace	ignored
maxInclusive	ignored
maxExclusive	ignored
minExclusive	ignored
minInclusive	ignored
totalDigits	precision in the copybook is derived from totalDigits
fractionDigits	scale in the copybook is derived from fractionDigits

Other Constraints

minOccurs	SOLA doesn't handle minOccurs automatically. It requires the programmer to specify "excludeif" during analysis
maxOccurs	converted to the occurs clause in the copybook, unless Custom, then all input and output arrays in Custom programs will get generated with maxOccurs="unbounded".

General Restrictions

- SOLA doesn't support RPC style WSDL. Document-literal is the only style that's supported.
- SOLA doesn't support non-SOAP bindings.
- SOLA discards array occurrences that exceed the value set during analysis. No warning is produced when data is discarded.
- The only XML schema to mainframe datatype transformations are based on the supported mainframe datatypes, such as character (XML normalization), binary (halfword and fullword) and packed decimal (numeric nibbles and sign nibble). An enhancement to Analysis has been made in release 6.3.4 so User can now change to any other compatible datatype during analysis and retain it, and if not compatible the datatype will be overridden with defaults. The default is for Numeric {short, int, long} based on precision, and the user can change to Decimal.
- Choice schema indicators are not supported.

Appendix B: Refreshing Templates in the SOLA STC

The SOLA STC uses "Templates" to store run-time meta data. A Template is an Assembler Data-Only Load Module. For performance reasons, the SOLA STC manages the loading and caching of Templates. Ordinarily this isn't an issue, but when you change a Template you need to refresh the Template in the SOLA STC.

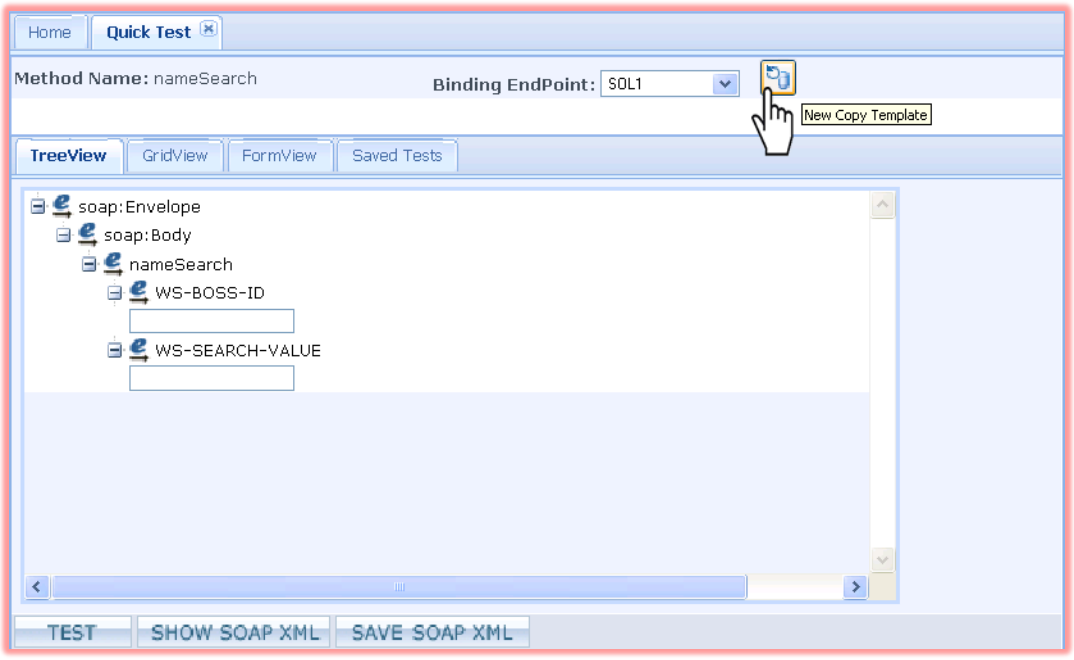
The SOLASTC provides two methods of refreshing a template:

- A manual method intended to be used by a programmer
- A web service method intended to be used for integration with a Change Management system.

Manually Refreshing a Template

CICS provides the ability to "NewCopy" a program with the CEMT transaction. SOLA IMS Container provides the same ability to "NewCopy" a template (the only user modifiable component hosted in the SOLA STC), but instead of providing a transaction SOLA provides a refresh button on the Quick Test pane.

After "**Analyzing**" a new method, right click on the Method and choose "Quick Test to bring up the Quick Test pane. In the upper right of the pane is a "refresh" button. Pressing this button refreshes the Template in the SOLA STC identified by the **Binding Endpoint**.



Refreshing a Template Using the Web Service Interface

SOLA provides integration points with your Change management system. One of those points is the "NewCopy" web service. By integrating the "NewCopy" service, you will be able to ensure that a template is available for use.

The following web service request can be executed against the SOLA STC to refresh the template:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ObjectService xmlns="http://project.ObjectFinder.x4mlsoa.com/SL/XMLPC804/">
         <Operation>select</Operation>
         <ObjectType>SOLAUtil</ObjectType>
         <Object objectType="SOLAUtil" operationType="newCopy"
programNm="[TemplateName]"/>
      </ObjectService>
   </soap:Body>
</soap:Envelope>
```

Replace [TemplateName] with the name of your template. The service can be executed from any web services client, including the SOLA Test Harness. Many customers use a SOLA Outbound Service from the SOLA Batch Container.

Appendix C: Overriding IMS Connect parameters on the soap:Header

IMS Connect parameters are specified at the Container Group level. See the Resource Manager User's Guide for information on specifying IMS Connect parameters for a Container Group.

The programmer has the ability to override IMS Connect parameters by specifying the parameters to be modified on the soap:Header of the input soap request. The following example shows how a programmer can override the IMS Connect exit to replace the value for IMSCIRMMsgId (which may have been specified on the Container Group) with a new value of *SAMPLE*.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
     <IMSConnectParm>
         <IMSCIRMMsgId>*SAMPLE*</IMSCIRMMsgId>
     </IMSConnectParm>
   </soap:Header>
   <soap:Body>
      <sampleSOLAIMSMain
xmlns="http://sampleSOLAIMSMain.sampleSOLAIMSMain.x4ml.soa.com/IM/SOLAIM01/SOA#IM01">
         <feet>5</feet>
         <inches>6</inches>
         <fahrenheit>77</fahrenheit>
      </sampleSOLAIMSMain>
   </soap:Body>
</soap:Envelope>
```

The allowable values that can be specified on the soap:Header are:

```
<IMSConnectParm>
   <IMSCCommitMode><0 or 1></IMSCCommitMode>
   <IMSCSyncLevel><none or confirm></IMSCSyncLevel>
   <IMSCDataStoreID>...</IMSCDataStoreID>
   <IMSCfqdn>...</IMSCfqdn>
   <IMSCIPaddress>...</IMSCIPaddress>
   <IMSCport>...</IMSCport>
   <IMSCTCPip>...</IMSCTCPip>
   <IMSCNumSessions>
   <IMSCIRMMsgId>8characters</IMSCIRMMsgId>
   <IMSCIRMClientID>8characters</IMSCIRMClientID>
   <IMSCIRMTermID>8characters</IMSCIRMTermID>
   <IMSCIRMUserID>8characters</IMSCIRMUserID>
   <IMSCIRMPasswd>8characters</IMSCIRMPasswd>
   <IMSCIRMGroup>8characters</IMSCIRMGroup>
   <IMSCIRMTran>8characters</IMSCIRMTran>
</IMSConnectParm>
```

CommitMode and SyncLevel Combinations supported by SOLA

1. CommitMode = 0 (Commit-then-Send) & SyncLevel = Confirm

   This is the default combination enforced by SOLA runtime.

   To specify this combination on the soap request pass the following in <soap:Header>

```
<soap:Header>
 <IMSConnectParm>
   <IMSCCommitMode>0</IMSCCommitMode>
   <IMSCSyncLevel>confirm</IMSCSyncLevel>
 </IMSConnectParm>
</soap:Header>
```

2. CommitMode = 1 (Send-then-Commit) & SyncLevel = None

   To specify this combination on the soap request pass the following in <soap:Header>

```
<soap:Header>
 <IMSConnectParm>
   <IMSCCommitMode>1</IMSCCommitMode>
   <IMSCSyncLevel>none</IMSCSyncLevel>
 </IMSConnectParm>
</soap:Header>
```

3. CommitMode = 1 (Send-then-Commit) & SyncLevel = Confirm

   To specify this combination on the soap request pass the following in <soap:Header>

```
<soap:Header>
 <IMSConnectParm>
   <IMSCCommitMode>1</IMSCCommitMode>
   <IMSCSyncLevel>confirm</IMSCSyncLevel>
 </IMSConnectParm>
</soap:Header>
```

Appendix D: Sample Custom Program

The following program contains comments that will help you use this sample program to construct your own custom programs for use with SOLA.

```
000100 IDENTIFICATION DIVISION.                                         12/12/97
000200 PROGRAM-ID. SOLACUEX.                                            TGWPC045
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500*-----------------------------------------------------------------*
000501* This is a sample 'Hello World' program which demonstrates the  *
000502* basics of writing a SOLA Custom Program (Version 2).           *
000503* This simple program accepts a 'Function' code from SOAP        *
000504* request and depending of the value of the code does one of the*
000505* following:                                                      *
000506*                                                                 *
000507* Function: HW   - Throws a SOAP Response 'Hello World'          *
000508* Function: HWF  - Throws a SOLA soap fault (return code = -1)   *
000509* Function: HWCF - Throws a Custom SOAP fault (return code = -2)*
000510*                                                                 *
000511* Following is a sample of what the SOAP request for this        *
000512* program would look like:                                       *
000513*                                                                 *
000514*<soap:Envelope                                                  *
000515*     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">    *
000516* <soap:Body>                                                     *
000517*  <helloWorld                                                    *
000518* xmlns="http://helloWorld.SolaExamples.x4ml.soa.com/CU/SOLACUEX/">
000519*   <Function>HW</Function>                                       *
000520*  </helloWorld>                                                  *
000521* </soap:Body>                                                    *
000522*</soap:Envelope>                                                *
000523*-----------------------------------------------------------------*
000530*-----------------------------------------------------------------*
000600 WORKING-STORAGE SECTION.
000700*-----------------------------------------------------------------*
000800
000900 01  WS-MISC-DISP-DATA.                                           00301003
001000     05  WS-CHANNEL-NM         PIC  X(16) VALUE 'SOLA-CUSTOM'.
001100     05  WS-STATUS-CONTAINER   PIC  X(16) VALUE 'SOLA-STATUS'.
001600     05  WS-FUNCTION           PIC  X(04) VALUE SPACES.       14800003
001610     05  WS-ABCODE             PIC  X(04)    VALUE SPACES.    14800003
001700     05  WS-RESP-EDIT          PIC ZZZZ9.                     00390799
001800     05  WS-RESP2-EDIT         PIC ZZZZ9.                     00390899
001900     05  WS-DOM-RC-EDIT        PIC ZZZZ9.                     00390899
002000     05  WS-SOAP-NS            PIC  X(41)    VALUE
002100         'http://schemas.xmlsoap.org/soap/envelope/'.
002200     05  WS-METHOD-NS.
002300         10  FILLER            PIC  X(31)    VALUE
002400             'http://helloWorld.SolaExamples.'.
002500         10  FILLER            PIC  X(23)    VALUE
002600             'x4mlsoa.com/CU/SOLACU24'.
002700
002800 01  WS-MISC-BINARY-DATA.                                         00301003
002900     05  WS-SUB                PIC S9(04)    VALUE +0 BINARY.   00301003
003000     05  WS-RESP               PIC S9(09) BINARY VALUE +0.      00520099
003100     05  WS-RESP2              PIC S9(09) BINARY VALUE +0.      00530099
003200     05  WS-STATUS-LEN         PIC S9(09) BINARY VALUE +0.      00530099
003300
```

```
003400 01  WS-MISC-POINTERS.                                              00090003
003500     05  WS-CONTAINER-PTR      USAGE IS POINTER VALUE NULL.        00053192
003600     05  WS-Dom-Ptr-Req        USAGE IS POINTER VALUE NULL.        00053192
003700                                                                    00270003
003800 01  WS-SWITCHES.                                                  01200099
003900     05  WS-TAG-FOUND-SW        PIC S9(04) BINARY VALUE +0.        01260099
004000         88  TAG-NOT-FOUND      VALUE +1.                          01270099
004100         88  TAG-FOUND          VALUE +2 +3 +4.                    01280099
004200         88  TAG-DATA-FOUND     VALUE +3.                          01290099
004300         88  NO-TAG-DATA        VALUE +4.                          01300099
004400                                                                    00270003
005100     COPY XMLDOMW1.
005200
005500*-----------------------------------------------------------------*
005600 LINKAGE SECTION.
005700*-----------------------------------------------------------------*
005800
005900 01  SOLA-Status-Area.
006000     COPY SOLACUV2.
006100
006200 01  SOAP-REQ-RESP              PIC  X(1000000).
006300
006400*-----------------------------------------------------------------*
006500 PROCEDURE DIVISION.
006600*-----------------------------------------------------------------*
006700     CONTINUE.
006800*-----------------------------------------------------------------*
006900 0000-MAINLINE.
007000*-----------------------------------------------------------------*
007100
007200     PERFORM 0010-GET-SOLA-STATUS-AREA
007300        THRU 0010-EXIT
007400
007500     PERFORM 0020-GET-SOAP-REQUEST
007600        THRU 0020-EXIT
007700
007800     PERFORM 0030-PARSE-SOAP-REQUEST
007900        THRU 0030-EXIT
008000
008100     PERFORM 0100-PROCESS-SERVICE-REQUEST
008200        THRU 0100-EXIT
008300
008400     PERFORM 0200-PUT-SOAP-RESPONSE
008500        THRU 0200-EXIT
008600
008700     PERFORM 0300-PUT-SOLA-STATUS-AREA
008800        THRU 0300-EXIT
008900
009000     GO TO 9999-RETURN
009100
009200     CONTINUE.
009300
009400 0000-EXIT.
009500     EXIT.
009600
009700*-----------------------------------------------------------------* 14978300
009800 0010-GET-SOLA-STATUS-AREA.
009900*-----------------------------------------------------------------* 14978300
010000                                                                        46
010100     EXEC CICS
010200        HANDLE ABEND
010300        LABEL(9999-HANDLE-ABEND)
010400     END-EXEC
```

```
010500
010600     MOVE LOW-VALUES TO WS-CHANNEL-NM                                03
010700
010800*    Get the channel associated with this transaction.
010900
011000     EXEC CICS ASSIGN
011100        CHANNEL(WS-CHANNEL-NM)                                       03
011200        RESP    (WS-RESP)
011300        RESP2   (WS-RESP2)
011400     END-EXEC
011500
011600     IF (WS-RESP NOT = +0) OR
011700        (WS-CHANNEL-NM = SPACES OR LOW-VALUES)                       03
011800        GO TO 9999-RETURN                                         02530003
011900     END-IF
012000
012100*    Get the SOLA Status container from that channel. The
012110*    contents of this container are described in COBOL copybook
012120*    SOLACUV2. This data structure will later be modified and
012130*    placed back into this container to communicated back to
012140*    SOLA the proper action to take (i.e. Throw a SOAP Response,
012150*    a SOLA Fault, or a Custom Fault.
012200
012300     MOVE 'SOLA-STATUS'              TO WS-STATUS-CONTAINER          03
012400     MOVE LENGTH OF SOLA-Status-Area TO WS-STATUS-LEN             02960000
012500
012600     EXEC CICS GET
012700        CONTAINER(WS-STATUS-CONTAINER)                               03
012800        SET      (WS-CONTAINER-PTR)
012900        FLENGTH  (WS-STATUS-LEN)
013000        RESP     (WS-RESP)
013100        RESP2    (WS-RESP2)
013200     END-EXEC
013300                                                                     46
013400     IF WS-RESP NOT = DFHRESP(NORMAL)
013500        GO TO 9999-RETURN                                         02530003
013600     END-IF
013700
013800     SET  ADDRESS OF SOLA-Status-Area TO WS-CONTAINER-PTR          02960000
013900     SET  CU-Return-Normal            TO TRUE
014000     MOVE SPACES                      TO CU-RETURN-MSG
014100                                                                     46
014200     CONTINUE.                                                    10750046
014300                                                                  10760046
014400 0010-Exit.                                                       10770046
014500     EXIT.                                                        10780046
014600                                                                  10790046
014700*-------------------------------------------------------------* 14978300
014800 0020-GET-SOAP-REQUEST.
014900*-------------------------------------------------------------* 14978300
015000                                                                     46
015010*    Retrieve the raw (unencrypted SOAP Request which was placed* 14978300
015011*    into this container by SOLA. Once retrieved, you may use   * 14978300
015012*    any means you wish to interogate its contents (we suggest  * 14978300
015013*    using the provided SOLA Soap Parser and API XMLPC112.      * 14978300
015020                                                                     46
015100     EXEC CICS GET
015200        CONTAINER(CU-REQ-CONTAINER)                                  03
015300        SET      (WS-CONTAINER-PTR)
015400        FLENGTH  (CU-REQUEST-LEN)
015500        RESP     (WS-RESP)
015600        RESP2    (WS-RESP2)
015700     END-EXEC
```

```
015800                                                                  46
015900     IF WS-RESP NOT = DFHRESP(NORMAL)
016000        SET  CU-Throw-Fault TO TRUE
016100        MOVE WS-RESP        TO WS-RESP-EDIT
016200        MOVE WS-RESP2       TO WS-RESP2-EDIT
016300        STRING 'Error getting SOAP Request container, RESP = '
016400               WS-RESP-EDIT
016500               ', RESP2 = '
016600               WS-RESP-EDIT
016700               DELIMITED BY SIZE
016800               INTO CU-RETURN-MSG
016900        END-STRING
017000        PERFORM 0300-PUT-SOLA-STATUS-AREA
017100           THRU 0300-EXIT
017200        GO TO 9999-RETURN                                  02530003
017300     END-IF
017400
017500     SET  ADDRESS OF SOAP-REQ-RESP TO WS-CONTAINER-PTR     02960000
017600                                                                 46
017700     CONTINUE.                                             10750046
017800                                                           10760046
017900 0020-Exit.                                                10770046
018000     EXIT.                                                 10780046
018100                                                           10790046
018200*-----------------------------------------------------------------* 14978300
018300 0030-PARSE-SOAP-REQUEST.
018400*-----------------------------------------------------------------* 14978300
018500
018510*    In this case the program will used the SOLA provide parser * 14978300
018511*    and DOM API (XMLPC112).  First the request must be parsed  * 14978300
018512*    into the DOM Tree as follows:                              * 14978300
018520
018600     SET  WS-DOM-PARSE    TO TRUE                          06203899
018700     SET  WS-DOM-HANDLE   TO NULL                          06203999
018800     MOVE +0              TO WS-DOM-CONTROL                06204099
018900     MOVE CU-REQUEST-LEN  TO WS-DOM-VALUE-LENGTH           06205099
019000                                                           06206099
019100     CALL WS-DOM-API USING WS-DOM-RC                       06208099
019200                         , WS-DOM-MSG                      06209099
019300                         , WS-DOM-HANDLE                   06210099
019400                         , WS-DOM-FUNCTION                 06220099
019500                         , WS-DOM-CONTROL                  06230099
019600                         , WS-DOM-TAG-NAME                 06240099
019700                         , SOAP-REQ-RESP                   02960000
019800                         , WS-DOM-VALUE-LENGTH             18880099
019900                                                           06270099
020000     IF WS-DOM-RC NOT = 0                                  06280099
020100        GO TO 9000-X-PROG-WITH-DOM-ERROR
020200     END-IF                                                06530199
020300                                                           06530275
020400     SET WS-Dom-Ptr-Req TO WS-DOM-HANDLE                   06530399
020500
020600     CONTINUE.                                             10750046
020700                                                           10760046
020800 0030-Exit.                                                10770046
020900     EXIT.                                                 10780046
021000                                                           10790046
021100*-----------------------------------------------------------------* 14978300
021200 0100-PROCESS-SERVICE-REQUEST.
021300*-----------------------------------------------------------------* 14978300
021400                                                           10790046
021410*    The following code shows an example of how to now use the  * 14978300
021420*    SOLA DOM API to interogate the contents of the SOAP        * 14978300
```

```
021430*    request. (See the sample SOAP request at the top of this   * 14978300
021431*    program)                                                   * 14978300
021440
021500     SET  WS-DOM-GET-ELEMENT-BYTAG TO TRUE                        16354099
021600     MOVE 'helloWorld'             TO WS-Dom-Parent               18850099
021700     MOVE 'Function'               TO WS-DOM-TAG-NAME
021800
021900     PERFORM 6000-Call-Dom-Api                                    18730099
022000        THRU 6000-EXIT                                            18730099
022100
022200     IF TAG-NOT-FOUND OR NO-TAG-DATA                              01270099
022300        SET  CU-Throw-Fault TO TRUE
022400        MOVE 'Function not found' TO CU-RETURN-MSG
022500        PERFORM 0300-PUT-SOLA-STATUS-AREA
022600           THRU 0300-EXIT
022700        GO TO 9999-RETURN                                         02530003
022800     END-IF
022900
023000     MOVE WS-DOM-VALUE(1:WS-DOM-VALUE-LENGTH) TO WS-FUNCTION      20660099
023100
023200     EVALUATE WS-FUNCTION
023300        WHEN 'HW'
023400           PERFORM 1000-SAY-HELLO-WORLD
023500              THRU 1000-EXIT
023600
023700        WHEN 'HWF'
023710*          This is the easiest way to send a SOAP fault from
023720*          a custom program.  Simply place the Fault message
023730*          you wish to comunicate in the CU-RETURN-MSG area
023740*          and set the CU-RETURN-CD to -1 (CU-Throw-Fault).
023750*          Then just place the status control block back into
023760*          the SOLA Status container and SOLA will do the rest.
023800           SET  CU-Throw-Fault            TO TRUE
023900           MOVE 'Sorry, World is offline' TO CU-RETURN-MSG
024000           PERFORM 0300-PUT-SOLA-STATUS-AREA
024100              THRU 0300-EXIT
024200           GO TO 9999-RETURN                                      02530003
024300
024400        WHEN 'HWCF'
024410*          This will demonstrate a technique for taking
024420*          complete control of the SOAP Fault that you will
024430*          send back to the requestor.
024500           GO TO 9100-THROW-HELLO-WORLD-FAULT
024700
024701        WHEN OTHER
024702           SET  CU-Throw-Fault          TO TRUE
024703           MOVE 'Invalid Function Code' TO CU-RETURN-MSG
024704           PERFORM 0300-PUT-SOLA-STATUS-AREA
024705              THRU 0300-EXIT
024706           GO TO 9999-RETURN                                      02530003
024707
024720
024800     END-EVALUATE
024900
025800     CONTINUE.                                                    10750046
025900                                                                  10760046
026000 0100-Exit.                                                       10770046
026100     EXIT.                                                        10780046
026200                                                                  10790046
038600*-------------------------------------------------------------* 14978300
038700 0200-PUT-SOAP-RESPONSE.
038800*-------------------------------------------------------------* 14978300
038900
```

```
038910*    This code will place the SOAP Response (built by this      * 14978300
038920*    program into the CU-RESP-CONTAINER. In this case the       * 14978300
038930*    CU-RETURN-CD should be set to zero (CU-RETURN-NORMAL).     * 14978300
038940*    It is this return code that will instruct SOLA to retrieve *
038941*    the contents of this container and deliver the SOAP        *
038942*    Response back to the requestor.                            *
038970
039000     MOVE 'SOAP-RESPONSE    ' TO CU-RESP-CONTAINER                    03
039100                                                                 05620099
039200     EXEC CICS PUT
039300        CONTAINER(CU-RESP-CONTAINER)                                 03
039400        From     (SOAP-Req-Resp)                                 05617200
039500        FLENGTH  (CU-RESPONSE-LEN)                               05620099
039600        RESP     (WS-RESP)
039700        RESP2    (WS-RESP2)
039800     END-EXEC
039900
040000     IF WS-RESP NOT = DFHRESP(NORMAL)
040100        SET  CU-Throw-Fault TO TRUE
040200        MOVE WS-RESP        TO WS-RESP-EDIT
040300        MOVE WS-RESP2       TO WS-RESP2-EDIT
040400        STRING 'Error putting SOAP Response container, RESP = '
040500               WS-RESP-EDIT
040600               ', RESP2 = '
040700               WS-RESP-EDIT
040800               DELIMITED BY SIZE
040900               INTO CU-RETURN-MSG
041000        END-STRING
041100        PERFORM 0300-PUT-SOLA-STATUS-AREA
041200           THRU 0300-EXIT
041300        GO TO 9999-RETURN                                        02530003
041400     END-IF
041500
041600     CONTINUE.                                                   10750046
041700                                                                 10760046
041800 0200-Exit.                                                      10770046
041900     EXIT.                                                       10780046
042000                                                                 10790046
042100*-----------------------------------------------------------------* 14978300
042200 0300-PUT-SOLA-STATUS-AREA.
042300*-----------------------------------------------------------------* 14978300
042400
042410*    The status control block (described in copybook SOLACUV2)  * 14978300
042420*    MUST be placed back into the CU-STATUS-CONTAINER to let    * 14978300
042430*    SOLA know what steps should be taken.                      * 14978300
042470
042500     EXEC CICS PUT
042600        CONTAINER(CU-STATUS-CONTAINER)                               03
042700        From     (SOLA-Status-Area)                                  00
042800        FLENGTH  (CU-STATUS-LEN)
042900        RESP     (WS-RESP)
043000        RESP2    (WS-RESP2)
043100     END-EXEC
043200
043300     IF WS-RESP NOT = DFHRESP(NORMAL)
043400        GO TO 9999-RETURN                                        02530003
043500     END-IF
043600
043700     CONTINUE.                                                   10750046
043800                                                                 10760046
043900 0300-Exit.                                                      10770046
044000     EXIT.                                                       10780046
044100                                                                 10790046
```

```
044110*-------------------------------------------------------------* 14978300
044120 1000-SAY-HELLO-WORLD.
044130*-------------------------------------------------------------* 14978300
044140                                                                20662899
044141*    Following is an example of how you could use the SOLA DOM  * 14978300
044142*    API to build a complete SOAP Response to be deliverd back  * 14978300
044143*    to the requestor.                                          * 14978300
044144
044150     SET  WS-DOM-CREATE-DOC TO TRUE                             18280099
044160     MOVE SPACES            TO WS-DOM-PARENT                    18300099
044170     MOVE +0                TO WS-Dom-Rc                        18310099
044180                             , WS-Dom-Control                   18320099
044190                             , WS-DOM-VALUE-LENGTH              18330099
044191                             , WS-DOM-PLACE-HOLDER
044192     MOVE 'soap:Envelope'   TO WS-DOM-TAG-NAME                  18392099
044193     SET  WS-Dom-Handle     TO NULL                             18340099
044194                                                                18350099
044195     PERFORM 6000-Call-Dom-Api                                  18730099
044196        THRU 6000-EXIT                                          18730099
044197                                                                18350099
044198     SET  WS-DOM-SET-ATTRIBUTE TO TRUE                          06939099
044199     MOVE 'soap:Envelope'      TO WS-Dom-Parent                 18350099
044200     MOVE 'xmlns:soap'         TO WS-DOM-TAG-NAME               18860099
044201     MOVE WS-SOAP-NS           TO WS-DOM-VALUE                  18870099
044202     MOVE LENGTH OF WS-SOAP-NS TO WS-DOM-VALUE-LENGTH           18330099
044203                                                                18350099
044204     PERFORM 6000-Call-Dom-Api                                  18730099
044205        THRU 6000-EXIT                                          18730099
044206                                                                18350099
044207     SET  WS-DOM-APPEND-CHILD  TO TRUE                          06937099
044208     MOVE 'soap:Envelope'      TO WS-Dom-Parent                 18350099
044209     MOVE 'soap:Body'          TO WS-DOM-TAG-NAME               18860099
044210     MOVE SPACES               TO WS-DOM-VALUE                  18870099
044211     MOVE +0                   TO WS-DOM-VALUE-LENGTH           18330099
044212                                                                18350099
044213     PERFORM 6000-Call-Dom-Api                                  18730099
044214        THRU 6000-EXIT                                          18730099
044215                                                                18350099
044216     SET  WS-DOM-APPEND-CHILD  TO TRUE                          06937099
044217     MOVE 'soap:Body'          TO WS-Dom-Parent                 18350099
044218     MOVE 'helloWorldResponse' TO WS-DOM-TAG-NAME               18860099
044219     MOVE SPACES               TO WS-DOM-VALUE                  18870099
044220     MOVE +0                   TO WS-DOM-VALUE-LENGTH           18330099
044221                                                                18350099
044222     PERFORM 6000-Call-Dom-Api                                  18730099
044223        THRU 6000-EXIT                                          18730099
044224                                                                18350099
044225     SET  WS-DOM-SET-ATTRIBUTE   TO TRUE                        06939099
044226     MOVE 'helloWorldResponse'   TO WS-Dom-Parent               18350099
044227     MOVE 'xmlns'                TO WS-DOM-TAG-NAME             18860099
044228     MOVE WS-METHOD-NS           TO WS-DOM-VALUE                18870099
044229     MOVE LENGTH OF WS-METHOD-NS TO WS-DOM-VALUE-LENGTH         18330099
044230                                                                18350099
044231     PERFORM 6000-Call-Dom-Api                                  18730099
044232        THRU 6000-EXIT                                          18730099
044233                                                                18350099
044234     SET  WS-DOM-APPEND-CHILD  TO TRUE                          06937099
044235     MOVE 'helloWorldResponse' TO WS-Dom-Parent                 18350099
044236     MOVE 'MessageToWorld'     TO WS-DOM-TAG-NAME               18860099
044237     MOVE 'Hello World'        TO WS-DOM-VALUE                  03728701
044238     MOVE +11                  TO WS-DOM-VALUE-LENGTH           18330099
044239                                                                18350099
044240     PERFORM 6000-Call-Dom-Api                                  18730099
```

```
044241        THRU 6000-EXIT                                            18730099
044242                                                                  18350099
044243*    Finalize will retrieve a copy of the completed SOAP Response.
044244                                                                  18350099
044245     SET  WS-DOM-FINALIZE TO TRUE                                 05614899
044246     MOVE +0              TO WS-DOM-VALUE-LENGTH                  05614999
044247                                                                  05615099
044248     PERFORM 6000-CALL-DOM-API                                    05616099
044249        THRU 6000-EXIT                                            05617099
044250                                                                  05617199
044251     SET  ADDRESS OF SOAP-Req-Resp TO WS-DOM-VALUE-PTR            05617200
044252     MOVE WS-DOM-VALUE-LENGTH      TO CU-RESPONSE-LEN             05620099
044253                                                                  05620099
044254     CONTINUE.                                                    10750046
044255                                                                  10760046
044256 1000-Exit.                                                       10770046
044257     EXIT.                                                        10780046
044258                                                                  10790046
044429*-------------------------------------------------------------* 18720099
044430 6000-Call-Dom-Api.                                              18730099
044440*-------------------------------------------------------------* 18740099
044500                                                                  18750099
044600     MOVE +0     TO WS-TAG-FOUND-SW                               18770099
044700                                                                  18800099
044800     CALL WS-DOM-API USING WS-Dom-Rc                              18810099
044900                         , WS-Dom-Msg                             18820099
045000                         , WS-DOM-HANDLE                          18830099
045100                         , WS-Dom-Function                        18840099
045200                         , WS-Dom-Parent                          18850099
045300                         , WS-DOM-TAG-NAME                        18860099
045400                         , WS-DOM-VALUE                           18870099
045500                         , WS-Dom-VALUE-LenGTH                    18880099
045600                                                                  18940099
045700     EVALUATE WS-DOM-RC                                           18950099
045800        WHEN +0                                                   18960099
045900           IF WS-Dom-VALUE-LenGTH > +0                            18970099
046000              SET Tag-Data-Found TO TRUE                          18980099
046100           ELSE                                                   18990099
046200              SET NO-TAG-DATA TO TRUE                             19000099
046300           END-IF                                                 19020099
046400                                                                  19030099
046500        WHEN +4                                                   19040099
046600           SET Tag-Not-Found TO TRUE                              19050099
046700                                                                  19060099
046800        WHEN OTHER                                                19070099
046900           GO TO 9000-X-PROG-WITH-DOM-ERROR
047000                                                                  19060099
047100     END-EVALUATE                                                 19270099
047200                                                                  19280099
047300     CONTINUE.                                                    19290099
047400                                                                  19300099
047500 6000-EXIT.                                                       19310099
047600     EXIT.                                                        19390099
047700                                                                  19400099
055800*-------------------------------------------------------------* 14713189
055900 9000-X-PROG-WITH-DOM-ERROR.
056000*-------------------------------------------------------------* 14713189
056100
056200     SET  CU-Throw-Fault TO TRUE
056300     MOVE WS-DOM-RC      TO WS-DOM-RC-EDIT                        19110099
056400
056500     PERFORM VARYING WS-SUB FROM LENGTH OF WS-DOM-MSG BY -1
056600       UNTIL WS-SUB = +1
```

```
056700          OR WS-DOM-MSG(WS-SUB:1) > SPACES
056800        CONTINUE
056900     END-PERFORM
057000
057100     STRING 'Error in DOM API, RC = '                              19160099
057200            WS-DOM-RC-EDIT                                         19170099
057300            ', DOM MSG = '                                         19190099
057400            WS-DOM-MSG(1:WS-SUB)                                   19190099
057500            DELIMITED BY SIZE                                      19200099
057600            INTO CU-RETURN-MSG
057700     END-STRING                                                    19240099
057800                                                                   20679999
057900     PERFORM 0300-PUT-SOLA-STATUS-AREA
058000        THRU 0300-EXIT
058100                                                                   20679999
058200     GO TO 9999-RETURN                                             02530003
058300                                                                   20679999
058400     CONTINUE.                                                     20680099
058500                                                                   20680199
058600 9000-Exit.                                                        20680299
058700     EXIT.                                                        20681099
058800                                                                   20681199
058810*-------------------------------------------------------------------* 18720099
058820 9100-THROW-HELLO-WORLD-FAULT.
058830*-------------------------------------------------------------------* 18720099
058831                                                                   20681199
058832*    Following is an example of how you could use the SOLA DOM  * 14978300
058833*    API to build a complete SOAP Fault to be deliverd back to  * 14978300
058834*    the requestor. Note that in order to instruct SOLA to send * 14978300
058835*    this fault document back to the request you will need to   * 14978300
058836*    set the CU-Return-Cd to -2 in the SOLA Status area.        * 14978300
058837
058854     SET  WS-DOM-CREATE-DOC TO TRUE                                18280099
058855     MOVE SPACES            TO WS-DOM-PARENT                       18300099
058860     MOVE +0                TO WS-Dom-Rc                           18310099
058870                            , WS-Dom-Control                       18320099
058880                            , WS-DOM-VALUE-LENGTH                  18330099
058890                            , WS-DOM-PLACE-HOLDER
058891     MOVE 'soap:Envelope'   TO WS-DOM-TAG-NAME                     18392099
058892     SET  WS-Dom-Handle     TO NULL                                18340099
058893                                                                   18350099
058894     PERFORM 6000-Call-Dom-Api                                     18730099
058895        THRU 6000-EXIT                                             18730099
058896                                                                   18350099
058897     SET  WS-DOM-SET-ATTRIBUTE TO TRUE                             06939099
058898     MOVE 'soap:Envelope'      TO WS-Dom-Parent                    18350099
058899     MOVE 'xmlns:soap'         TO WS-DOM-TAG-NAME                  18860099
058900     MOVE WS-SOAP-NS           TO WS-DOM-VALUE                     18870099
058901     MOVE LENGTH OF WS-SOAP-NS TO WS-DOM-VALUE-LENGTH              18330099
058902                                                                   18350099
058903     PERFORM 6000-Call-Dom-Api                                     18730099
058904        THRU 6000-EXIT                                             18730099
058905                                                                   18350099
058906     SET  WS-DOM-APPEND-CHILD  TO TRUE                             06937099
058907     MOVE 'soap:Envelope'      TO WS-Dom-Parent                    18350099
058908     MOVE 'soap:Body'          TO WS-DOM-TAG-NAME                  18860099
058909     MOVE SPACES               TO WS-DOM-VALUE                     18870099
058910     MOVE +0                   TO WS-DOM-VALUE-LENGTH              18330099
058911                                                                   18350099
058912     PERFORM 6000-Call-Dom-Api                                     18730099
058913        THRU 6000-EXIT                                             18730099
058914                                                                   18350099
058915     SET  WS-DOM-APPEND-CHILD  TO TRUE                             06937099
```

```
058916     MOVE 'soap:Body'          TO WS-Dom-Parent                    18350099
058917     MOVE 'soap:Fault'         TO WS-DOM-TAG-NAME                  18860099
058918     MOVE SPACES               TO WS-DOM-VALUE                     18870099
058919     MOVE +0                   TO WS-DOM-VALUE-LENGTH              18330099
058920                                                                   18350099
058921     PERFORM 6000-Call-Dom-Api                                     18730099
058922        THRU 6000-EXIT                                             18730099
058923                                                                   18350099
058924     SET  WS-DOM-APPEND-CHILD  TO TRUE                             06937099
058925     MOVE 'soap:Fault'         TO WS-Dom-Parent                    18350099
058926     MOVE 'faultcode'          TO WS-DOM-TAG-NAME                  18860099
058927     MOVE 'soap:Client'        TO WS-DOM-VALUE                     03728701
058928     MOVE +11                  TO WS-DOM-VALUE-LENGTH              18330099
058929                                                                   18350099
058930     PERFORM 6000-Call-Dom-Api                                     18730099
058931        THRU 6000-EXIT                                             18730099
058932                                                                   18350099
058933     SET  WS-DOM-APPEND-CHILD      TO TRUE                         06937099
058934     MOVE 'soap:Fault'             TO WS-Dom-Parent                18350099
058935     MOVE 'faultstring'            TO WS-DOM-TAG-NAME              18860099
058936     MOVE 'ERROR101-World Problem' TO WS-DOM-VALUE                 03728701
058937     MOVE +24                      TO WS-DOM-VALUE-LENGTH          18330099
058938                                                                   18350099
058939     PERFORM 6000-Call-Dom-Api                                     18730099
058940        THRU 6000-EXIT                                             18730099
058941                                                                   18350099
058942                                                                   05620099
058943     SET  WS-DOM-APPEND-CHILD        TO TRUE                       06937099
058944     MOVE 'soap:Fault'               TO WS-Dom-Parent              18350099
058945     MOVE 'detail'                   TO WS-DOM-TAG-NAME            18860099
058946     MOVE SPACES                     TO WS-DOM-VALUE               03728701
058947     MOVE +0                         TO WS-DOM-VALUE-LENGTH        18330099
058948                                                                   18350099
058949     PERFORM 6000-Call-Dom-Api                                     18730099
058950        THRU 6000-EXIT                                             18730099
058951                                                                   18350099
058952                                                                   18350099
058953     SET  WS-DOM-APPEND-CHILD                  TO TRUE             06937099
058954     MOVE 'detail'                             TO WS-Dom-Parent    18350099
058955     MOVE 'e:message'                          TO WS-DOM-TAG-NAME  18860099
058956     MOVE 'Sorry, World is offline (My Fault)' TO WS-DOM-VALUE     03728701
058958     MOVE +34 TO WS-DOM-VALUE-LENGTH                               18330099
058959                                                                   18350099
058960     PERFORM 6000-Call-Dom-Api                                     18730099
058961        THRU 6000-EXIT                                             18730099
058962                                                                   18350099
058963     SET  WS-DOM-SET-ATTRIBUTE               TO TRUE               06939099
058964     MOVE 'e:message'                        TO WS-Dom-Parent      18350099
058965     MOVE 'xmlns:e'                          TO WS-DOM-TAG-NAME    18860099
058966     MOVE 'http://www.dsd.ml.com/x4ml/fault' TO WS-DOM-VALUE       18870099
058967     MOVE +33 TO WS-DOM-VALUE-LENGTH                               18330099
058968                                                                   18350099
058969     PERFORM 6000-Call-Dom-Api                                     18730099
058970        THRU 6000-EXIT                                             18730099
058971                                                                   18350099
058972*    Finalize will retrieve a copy of the completed SOAP Fault.
058973                                                                   18350099
058974     SET  WS-DOM-FINALIZE TO TRUE                                  05614899
058975     MOVE +0              TO WS-DOM-VALUE-LENGTH                   05614999
058976                                                                   05615099
058977     PERFORM 6000-CALL-DOM-API                                     05616099
058978        THRU 6000-EXIT                                             05617099
058979                                                                   05617199
```

```
058980     SET  ADDRESS OF SOAP-Req-Resp TO WS-DOM-VALUE-PTR            05617200
058982     SET  CU-CUSTOM-FAULT          TO TRUE
058983     MOVE 'SOAP-FAULT'             TO CU-FAULT-CONTAINER                03
058984     MOVE WS-DOM-VALUE-LENGTH      TO CU-FAULT-LEN                05620099
058985                                                                  20681199
058986*    Place the completed fault into the CU-FAULT-CONTAINER.
058987                                                                  20681199
059006     EXEC CICS PUT
059007        CONTAINER(CU-FAULT-CONTAINER)                                  03
059011        From     (SOAP-Req-Resp)                               05617200
059013        FLENGTH  (CU-FAULT-LEN)                                05620099
059014        RESP     (WS-RESP)
059015        RESP2    (WS-RESP2)
059016     END-EXEC
059017
059018     IF WS-RESP NOT = DFHRESP(NORMAL)
059019        SET  CU-Throw-Fault TO TRUE
059020        MOVE WS-RESP        TO WS-RESP-EDIT
059021        MOVE WS-RESP2       TO WS-RESP2-EDIT
059022        STRING 'Error putting SOAP Fault container, RESP = '
059023               WS-RESP-EDIT
059024               ', RESP2 = '
059025               WS-RESP-EDIT
059026               DELIMITED BY SIZE
059027               INTO CU-RETURN-MSG
059028        END-STRING
059032     END-IF
059033
059034*    Now place the proper status into the CU-STATUS-CONTAINER
059035
059036     PERFORM 0300-PUT-SOLA-STATUS-AREA
059037        THRU 0300-EXIT
059038                                                                  20679999
059039     GO TO 9999-RETURN                                            02530003
059040                                                                  20679999
059041     CONTINUE.                                                    10750046
059042                                                                  10760046
059046 9100-Exit.                                                       10770046
059047     EXIT.                                                        10780046
059048                                                                  10790046
059182*-------------------------------------------------------------* 14713189
059183 9999-HANDLE-ABEND.                                              14720003
059190*-------------------------------------------------------------* 14730003
059200                                                                  14740003
059300     EXEC CICS HANDLE ABEND                                       14750003
059400        CANCEL                                                    14760003
059500     END-EXEC.                                                    14770003
059600                                                                  14780003
059700     EXEC CICS ASSIGN                                             14790003
059800        ABCODE (WS-ABCODE)                                        14800003
059900     END-EXEC                                                     14810003
060000                                                                  14820003
060100     SET  CU-Throw-Fault TO TRUE
060200     MOVE WS-RESP        TO WS-RESP-EDIT
060300     MOVE WS-RESP2       TO WS-RESP2-EDIT
060400     STRING 'Abend in program SOLACU24, Code = '
060500            WS-ABCODE                                             14800003
060600            DELIMITED BY SIZE
060700            INTO CU-RETURN-MSG
060800     END-STRING
060900                                                                  02750003
061000     PERFORM 0300-PUT-SOLA-STATUS-AREA
061100        THRU 0300-EXIT
```

```
061200                                                                  02750003
061300     GO TO 9999-RETURN                                            02530003
061400                                                                  14880003
061500     CONTINUE.                                                    14890003
061600                                                                  14900003
061700 9999-HANDLE-ABEND-X.                                             14920003
061800     EXIT.                                                        14930003
061900                                                                  14940003
062000*-------------------------------------------------------------* 14950003
062100 9999-RETURN.                                                     14960003
062200*-------------------------------------------------------------* 14970003
062300                                                                  14980003
062400     EXEC CICS                                                    14990003
062500        RETURN                                                    15000003
062600     END-EXEC.                                                    15010003
```