Avaya Interaction Center

Avaya Agent Integration

Release 7.3.x
October 2015

Contents

Chapter 1: Introduction

Purpose

The purpose of this guide is to provide detailed information about integrating the Avaya Agent framework into the Windows system that agent uses.

Intended audience

This guide is intended primarily for an administrator who is authorized to integrate the Avaya Agent framework for an agent to use.

Reason for reissue

Following section is added in the document:

- Unified Agent Directory API on page 59

Related resources

Documentation

See the following related documents at http://support.avaya.com.

Title	Use this document to:	Audience
Avaya Interaction Center and Avaya Operational Analyst Overview and Specification	get information about the new features and enhancements in Avaya Interaction Center.	Sales Engineers Supervisors Business Partners Customers
IC Installation Planning and Prerequisites	get information about the planning and third-party software required to deploy an Avaya Interaction Center system.	Business partners Customers Implementation engineers
IC Installation and Configuration	get information about how to install and configure an out-of-the-box Avaya Interaction Center.	Sales Engineers Supervisors Business Partners Customers
Agent User Guide	get agent-related information about Avaya Interaction Center Agent.	Business partners Customers Agents Supervisors
Avaya Agent Web Client	get information about Avaya Agent Web Client.	Business partners Customers Agents Supervisors

Title	Use this document to:	Audience
IC Administration Guide	get information about Avaya Interaction Center (Avaya IC). This guide describes domain and server administration using Avaya IC Manager.	Sales Engineers Supervisors Administrators Business Partners Customers
Agent Web Client Customization	get information about how to customize Avaya Agent Web Client.	Supervisors Administrators Business Partners Administrators

Finding documents on the Avaya Support website

Use this procedure to find product documentation on the Avaya Support website.

1. Use a browser to navigate to the Avaya Support website at http://support.avaya.com.
2. At the top of the screen, enter your username and password and click **Login**.
3. Click **Documents**.
4. In the **Enter your Product Here** search box, type the product name and then select the product from the drop-down list.
5. If there is more than one release, select the appropriate release number from the **Choose Release** drop-down list.
6. Use the **Content Type** filter on the left to select the type of document you are looking for, or click Select All to see a list of all available documents.

   For example, if you are looking for user guides, select **User Guides** in the **Content Type** filter. Only documents in the selected category will appear in the list of documents.
7. Click **Enter**.

Training

The following courses are available on the Avaya Learning website at www.avaya-learning.com. In the **Search** field, enter the course code, and click **Go to search** for the course.

Course Code	Course Title
ATC01175WEN	IC and OA Overview
ATC01176IEN	Interaction Center Administration and Configuration

Course Code	Course Title
AUCC100010695	IC-Siebel Integration
ATC100011017	IC Siebel Integration, Installation and Troubleshooting

Viewing Avaya Mentor videos

Avaya Mentor videos provide technical content on how to install, configure, and troubleshoot Avaya products.

Videos are available on the Avaya Support website, listed under the video document type, and on the Avaya-run channel on YouTube.

- To find videos on the Avaya Support website, go to http://support.avaya.com and perform one of the following actions:
  - In Search, type Avaya Mentor Videos to see a list of the available videos.
  - In Search, type the product name. On the Search Results page, select Video in the Content Type column on the left.
- To find the Avaya Mentor videos on YouTube, go to www.youtube.com/AvayaMentor and perform one of the following actions:
  - Enter a key word or key words in the Search Channel to search for a specific product or topic.
  - Scroll down Playlists, and click the name of a topic to see the available list of videos posted on the website.

Note:
Videos are not available for all products.

Support

Go to the Avaya Support website at http://support.avaya.com for the most up-to-date documentation, product notices, and knowledge articles. You can also search for release notes, downloads, and resolutions to issues. Use the online service request system to create a service request. Chat with live agents to get answers to questions, or request an agent to connect you to a support team if an issue requires additional expertise.

Chapter 2: Overview

The Avaya Agent is a framework that can host OLE controls for an agent to use.

The framework is similar to the Microsoft Windows taskbar, consisting of one or more QFrames (windows) that can be anchored to the left, right, top, or bottom of the agent's screen. These frames contain tabbed QPanes on which the OLE controls reside.

When agents launch Avaya Agent, their desktop is resized and Windows prevents other applications from using the area occupied by the Avaya Agent frames. The rest of the agent's desktop is available for other applications, such as Microsoft Word.

This section includes the following sections:

Integration tasks

As the system integrator for Avaya Agent, you define the agent's work environment by deciding what is displayed.

You need to determine:

- What controls you want displayed.
- What logical groups can be made from the controls—these groups become your panes.
- How many Avaya Agent QFrames you need and where they will appear.
- What QPanes you want in each QFrame.
- The size and position of the Avaya Agent QFrame(s), QPane(s), and controls.
- How the controls interact with each other and with any application running in the desktop area, and which IC Scripts you need to run to make that interaction happen.

Avaya Agent's layout is written in an XML (eXtensible Markup Language) specification file called the CDL (Console Definition Language) file that you modify and save to the database. When an agent logs in, the system accesses the stored layout and sets up the agent's machine accordingly.

When you install the Avaya Agent design layouts, you select which configuration is the closest match to what you intend to implement at your company. Then you use the default CDL file as a starting point. For more information, see The format of the CDL file on page 20.

The interactions between controls, and between a control and an application, are governed by a set of IC Scripts you write or modify using Visual Basic for Applications (VBA) based IC Script methods provided by Avaya. For more information, see *IC Scripts Language Reference*.

IC Scripts overview

IC Scripts are VBA based subroutines that can be run either explicitly or when an event is raised.

Events can be raised by:

- Controls.
- Framework events such as pane activation or deactivation. For more information, see *IC Scripts Language Reference*.
- Mouse clicks or key clicks.

These subroutines can do almost anything, from displaying an alert for the agent to saving information into the database. For more information, see *IC Scripts Language Reference*.

Information on how to change the behavior of Avaya Agent by writing IC Scripts that handle the available Integration Hooks is provided in Customizing Avaya Agent on page 19.

Component overview

Components are made up of a control (or group of controls) and the corresponding IC Scripts used to integrate the control(s) into Avaya Agent. Out-of-the-box, Avaya Agent has several different types of components. This section lists the types of components with a brief description.

Core Services: This component is the basis for the Avaya IC environment. It must be in place before any Media Channel can be integrated.

Media Channels: These are components used for handling media contacts in Avaya Agent. The following are included in Avaya Agent:

- Voice: Voice is provided through the Softphone, which is an electronic interface to an agent's phone that controls all the standard telephony functions. It includes:

  - Telephony aware buttons that support various operations such as answer call, hang up, and make call.
  - Informational fields that can display the agent or phone state.
  - A call list that shows all active lines.
  - A dial directory.
- Email: Allows Customer Service Representatives (CSRs) to view and respond to email messages that come into the contact center.
- Chat: Allows CSRs to interact with contacts that come into the contact center via Chat through the Web Agent Client.

Contact Viewing: These components can be used in conjunction with one or more Media components. The following are the Contact Viewing components included in Avaya Agent:

- EDU Viewer: A GUI interface to the information contained in an Electronic Data Unit (EDU) that is associated with every contact.
- Contact History Browser, Contact History Filter, and Active Contact Viewer: Allows CSRs to view all historical contacts for a given contact.

Contact Wrapup: This component is used for wrapping up contacts across any media:

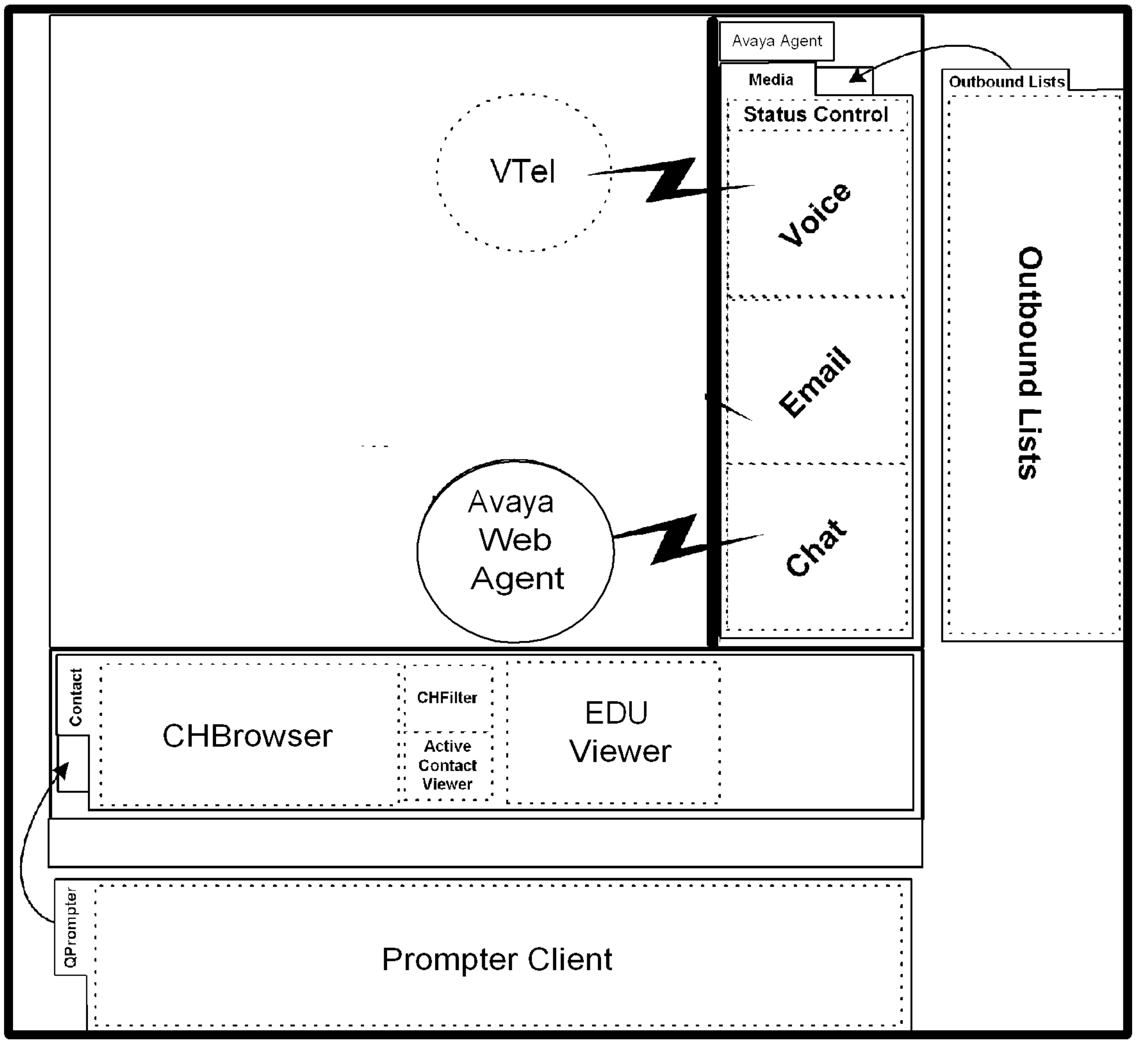
- Wrapup Dialog Control. A dialog box that appears when a contact is completed that requests wrapup information from the agent.

Standalone: These are components which can be used by themselves or with others. The following are included in Avaya Agent:

- PrompterClient: PrompterClient is a component used for running Prompter Flows in Avaya Agent.

Default component layout

The following drawing shows the location of components in the out-of-the-box Avaya Agent layout:



Customization steps

The Avaya Agent interface developer should complete the following tasks to produce a functioning contact center application:

1. Design the general flow of the interface based on the components described in this manual.
2. Mock up the interface: location, customizable components and application.
3. Review the `avaya_agent_en.cdl` file found in `*IC_INSTALL_DIR*\IC73\design\qconsole` for items to alter. This file is created during installation. Modify or remove items based on your design.
4. Modify those IC Scripts which interact with the components you have customized. You would use IC Scripts to perform activities triggered by events in the Avaya Agent interface. Refer to *IC Scripts Language Reference* for more information.
5. Design Media workflows and Prompter scripts using Workflow Designer. In general, you use these scripts to enforce business rules. You may use workflows to control the flow of contacts entering your contact center. Prompter scripts are used to prompt and direct responses from agents in different scenarios. For more information, refer to the *Avaya IC Media Workflow Reference* and *Agent Script Workflow Reference.*
6. Save both modified IC Scripts and the `avaya_agent_en.cdl` file to the application database via Database Designer. For more information, refer to *IC Database Designer Application Reference*.

Chapter 3: Avaya Agent basics

This chapter describes the basic elements of using Avaya Agent. It includes how to run the application, some of the fundamental keystrokes required for navigating around Avaya Agent, how to specify Avaya Agent properties, and some of the language implications with Avaya Agent.

This section includes the following sections:

Running Avaya Agent

When you have created your CDL file(s) and IC Scripts, you save them to the database as described in the *IC Database Designer Application Reference*. When they are uploaded, you can have your agents access them by setting up a shortcut for each agent using the command line arguments described below. Note that shortcut keys only work when Avaya Agent has the focus, not when the customer application is focused.

When you set up shortcuts for your agents, you can specify the following command line arguments:

Parameter	Value	Description
-d[irectory]	"file_location"	Directory where the temporary files that are downloaded from the database are kept. If this entry is not mentioned the files go to the system temp directory.
-n[ame]	"datasource_name"	Name of the datasource mentioned in IC Manager. This information is put in the Datasource field of the login dialog.
-u[ser]	"loginname"	Login name of the user. If this parameter is specified along with the -p option, Avaya Agent will login automatically.

Parameter	Value	Description
-p[assword]	"password"	Login password. If this parameter is specified along with the -u option, Avaya Agent will login automatically.
-layout	"layout_name"	The database name of the Avaya Agent layout spec to use when the agent logs into Avaya Agent. This name may be case sensitive depending on the database you are logging into. This parameter overrides the entry for the agent property (Under “Agent/Desktop” Layout property has to be set)

Note:
For a description of ADL files, refer to *IC Database Designer Application Reference*.

Keyboard navigation

In addition to any keyboard shortcuts you have defined for a pane, Avaya Agent includes these standard shortcut keys:

Key	Function
F6	Navigate between QFrames.
<ctrl Tab>	Navigate between tabbed QFrames, and the Avaya Agent button if there is one in the currently active pane.
Tab, <shift Tab>	Navigate between controls.

Specifying Avaya Agent properties

This section lets you specify name/value pairs that can be used by the IC Scripts associated with Avaya Agent.

The following table describes some of the IC Manager properties that are used by Avaya Agent. For more information on setting properties, see *IC Administration Guide*.

Section	**Description**
Agent/Desktop	This section contains the Agent's Desktop settings applicable to Avaya Agent.
Agent/Desktop/BlendingMode	This section contains properties for defining how Blending Mode works in the Agent’s Desktop.
Agent/Desktop/ContactSuspension	This section contains properties for defining how Contact Suspension works in the Agent's Desktop.
Agent/Desktop/Directory	This section contains properties for defining the behavior of the Directory in the Agent's Desktop.
Agent/Desktop/Directory/SkillProficiency	This section contains Skill Proficiency-specific properties of the Directory in the Agent's Desktop.
Agent/Desktop/Directory/Voice	This section contains Voice-specific properties of the Directory in the Agent's Desktop.
Agent/Desktop/Email	This section contains email specific properties of the agent's desktop.
Agent/Desktop/Email/AlertInfo/REQ	This section contains properties for to the Alert Info to be set for Emails sent to a Customer requiring more info from the Agent's Desktop.
Agent/Desktop/Email/AlertInfo/SME	This section contains properties for the Alert Info to be set for emails sent to a Subject Matter Expert from the Agent's Desktop.
Agent/Desktop/Layout	This section contains properties for associating a Console layout to a workgroup/agent
Agent/Desktop/Prompter	This section contains properties for defining the behavior of the Prompter in the Agent's Desktop.
Agent/Desktop/ScreenPop	This section contains properties for defining how Screen Pops work in the Agent's Desktop.
Agent/Desktop/Softphone	This section contains properties for defining the behavior of the Softphone in the Agent's Desktop.

Section	Description
Agent/Desktop/WAC	This section contains properties for defining the behavior of the Web Agent in the Agent's Desktop.
Agent/Desktop/WrapUpDialog	This section contains properties for defining the behavior of the WrapUp Dialog in the Agent's Desktop.

Language implications

If you modified the standard CDL file to customize your Avaya Agent layout, this section describes how to specify language properties in this file.

Avaya IC provides standard CDL files to enable language properties. They are named Avaya Agent with a language suffix appended to the filename. Some examples of these files are:

- avaya_agent_en.cdl = English
- avaya_agent_jp.cdl = Japanese
- avaya_agent_fr.cdl = French

You can use the syntax in these files as examples as you change your customized CDL to specify language properties.

For this example, use "avaya_agent_yours.cdl" as the name of your customized CDL file.

To specify language properties:

1. Make a copy of your customized CDL file in the same directory.
2. Append a language suffix to the filename. For example: avaya_agent_yours_en.cdl. Refer to the *IC Administration Guide* for a complete list of supported language suffixes.
3. Open the CDL file and edit the <QConsole name = > section to read the exact name of your customized CDL file.
   For example: <QConsole name = avaya_agent_yours_en.cdl>
4. Save the changes to the file.
5. Repeat steps 1 through 4 for each language you want to support changing the language suffix to the filename and the file text as appropriate.

Avaya IC will select the appropriate CDL file based on the setting of the UILanguage property in IC Manager.

Chapter 4: Customizing Avaya Agent

This chapter discusses the various ways you can customize Avaya Agent to meet your specific business requirements. Typically, customizations to Avaya Agent fall into one of the following categories: Layout, Behavior, and Database.

This section includes the following topics:

- Customizing Avaya Agent layout on page 20
- Customizing Avaya Agent behavior on page 36
- Connecting to an external database on page 46

Customizing Avaya Agent layout

As discussed in Overview on page 9, Avaya Agent consists of one or more frames whose size, position, and contents you specify using a CDL file written in XML. In those frames, you can include one or more panes, and in each of those panes can embed one or more controls.

Note:
While customizing the Avaya Agent interface, if you add or remove the UI controls or change the position of the UI controls, ensure that the UI controls are visible on the user interface and do not overlap.

During the Avaya Agent designer installation, the default CDL file is saved in the `IC_INSTALL_DIR\IC73\design\QConsole\avaya_agent_en.cdl` directory.

You can edit CDL file using any text or XML editor, and then save it to the database using Database Designer. (For details, refer to *IC Database Designer Application Reference*.)

The format of the CDL file

The CDL file is built from a series of tags. Some tags affect the application as a whole, while others apply to just one pane or control. The file has a nested structure; each section starts with a beginning tag and ends with a closing tag.

The basic structure looks like this:

```
Start QConsole
    Login options
    Avaya Agent properties
    IC Script Files
    Framework IC Scripts
    Frame 1
      Pane 1 in Frame 1
        IC Scripts for a pane
        Controls in the pane
          IC Scripts for the control
        End Controls
        End Pane 1
      Pane 2 in Frame 1
        IC Scripts for a pane
        Controls in the pane
          IC Scripts for the control
        End Controls
      End Pane 2
    End Frame 1
    Frame 2
      ... Pane, IC Script, and Control definitions ...
    End Frame 2
End QConsole
```

The available tags are described in this chapter. To see examples of how the out-of-the-box version of Avaya Agent looks and functions when you put the entire system together, refer to *Avaya Agent User Guide*.

All CDL tags have the format `<tagname ...attributes...> [optional intermediate tags] </tagname>` to show the start and end points for each tag. If a tag is only one line long, you can also use the format `<tagname ...attributes... />` to designate the start and end of a tag instead of using the full `</tagname>`.

Setting global options

The first section in the CDL file lets you set the global options for Avaya Agent.

<QConsole> ... </QConsole>

Description

The QConsole tag lets you set global Avaya Agent attributes.

Syntax

`<`**`QConsole`**` Name="`*Name*`" Version="`*Version*`" Description="`*Description*`">`

```
    ...rest of the QConsole definition statements...
</QConsole>
```

Attribute	Value	Description
Name	Any text string	The case-sensitive name of the CDL layout in the database. Each layout must have a unique name.
Version	Any text string	The version number so you can keep track of changes.
Description	Any text string	A description of the layout that should indicate what the layout is used for.

Example

```
<QConsole Name="avaya_agent_en" Version="7.3" Description="Default Avaya
Agent Layout Spec in English">
```

Specifying the login dialog

This section lets you specify which controls require login information and what information they need. It is built from a series of tags that get increasingly more specific, and there is a cumulative example after the QField tag to show what they look like when put together.

For a detailed description of the login process, see Avaya Agent initialization and exit hooks on page 52.

<QLogin> ... </QLogin>

Description

The QLogin tag lets you specify which components need login information.

Syntax

```
<QLogin>
    ...rest of the QLogin section statements...
</QLogin>
```

<QComponentDictionary> ... </QComponentDictionary>

Description

The QComponentDictionary tag is a wrapper around all the components that require login information.

Syntax

```
<QLogin>
    <QComponentDictionary>
...QComponent definition statements...
    </QComponentDictionary>
</QLogin>
```

<QComponent> ... </QComponent>

Description

The QComponent tag provides a tab name that will be used for a specific component and becomes a wrapper around the login fields. You may add QField tags within this tag; these fields will be added to the first Avaya Agent Login tab.

Syntax

```
<QLogin>
    <QComponentDictionary>
       <QComponent Name="Name" Visible="Visible">
       ...field definitions...
    </QComponent>
</QComponentDictionary>
</QLogin>
```

Attribute	Value	Description
Name	Any text string	The name of the component. This name is displayed as the tab name in the Avaya Agent Login dialog box.
Visible	“True” or “False”	Whether the component is visible or not. Default: True.
Label	Any text string	This is the name that is displayed in the tab of the login.

<QField> ... </QField>

Description

The QField tag lets you specify what login fields to display for a specific component.

Syntax

```
<QLogin>
    <QComponentDictionary>
<QComponent Name="sName">
```

```
    <QField Label="Label" Space="Space" Password="Password" />
</QComponent>
    </QComponentDictionary>
</QLogin>
```

Attribute	Value	Description
Label	Any text string	The field label.
Space	Any integer	The number of blank lines to leave between the label and the input field. For example, to have the input field on the same line as the label, you would use `Space="0"` (zero). To have it on the following line, use `Space="1"` (one). Default: `Space="0"`
Password	"True" or "False"	Whether this field is a password. If this is set to True, Avaya Agent replaces the user's input with asterisks (*). Default: False.

Example

```
<QLogin>
    <QComponentDictionary>
<QComponent Name="Application">
    <QField Label="Login ID:" Space="0" Password="FALSE" />
    <QField Label="Password:" Space="0" Password="TRUE" />
</QComponent>
    </QComponentDictionary>
</QLogin>
```

Specifying Avaya Agent properties

This section lets you specify name/value pairs that can be used by the IC Scripts associated with Avaya Agent.

<QPropertyDictionary> ... </QPropertyDictionary>

Description

The QPropertyDictionary tag begins and ends the name/value pair definition section.

Syntax

```
<QPropertyDictionary>
    ...name/value statements...
</QPropertyDictionary>
```

<QSection> ... </QSection>

Description

The QSection tag lists the name/value pair (or pairs) you want to set for a given section. (You can think of each QSection as the equivalent of an application Preferences section, where functionally-related properties are grouped together under a common header.)

Syntax

```
<QPropertyDictionary>
   <QSection Name="Name" sPairName="Value" [sPairName="Value"
    sPairName="Value" ... sPairName="Value"]>
   </QSection>
</QPropertyDictionary>
```

Attribute	Value	Description
Name	Any text string	The name of the section whose name/ value pairs you want to set.
PairName	Any text string	The name portion of the name/value pair.
Value	Any text string	The value portion of the name/value pair.

Example

```
<QPropertyDictionary>
    <QSection Name="General" HasApp="False" HasSoftphone="False"/>
    <QSection Name="Application" CommandLine=""/>
    <QSection Name="Softphone" ACD="False"/>
</QPropertyDictionary>
```

Specifying framework IC Scripts

The next two sections are the:

- QScript File Dictionary, which is the collection of files in which the Avaya Agent IC Scripts are stored.

- QScript Dictionary, which specifies what IC Scripts are to be run at the main Avaya Agent hook points. For details, refer to *IC Scripts Language Reference*.

<QScriptFileDictionary> ... </QScriptFileDictionary>

Description

The QScriptFileDictionary tag begins and ends the list of files that contain the Avaya Agent IC Scripts.

Syntax

```
<QScriptFileDictionary>
    ...IC Script files...
</QScriptFileDictionary>
```

<QScriptFile> ... </QScriptFile>

Description

The QScriptFile tag lists gives the name of an IC Script and specifies what file it is in. The file attribute is that file name you open in Database Designer and save to the database. You should have a <QScriptFile> definition for each IC Script that your Avaya Agent application will run.

Syntax

```
<QScriptFileDictionary>
    <QScriptFile Name="Name" File="File">
    </QScriptFile>
</QScriptFileDictionary>
```

Attribute	Value	Description
Name	Any text string	The name of the IC Script.
File	Any text string	The file name in which the IC Script resides. This can be a fully qualified file name, or a name relative to the directory in which the CDL layout (the one that you open in Database Designer and compiled) is stored. Default is relative to the CDL layout directory.

Example

```
<QScriptFileDictionary>
    <QScriptFile Name="QConsole_BeforeLogin" File="qconsole.qsc"/>
    <QScriptFile Name="QConsole_Login" File="qconsole.qsc"/>
    <QScriptFile Name="QConsole_AfterLogin" File="qconsole.qsc"/>
    <QScriptFile Name="QConsole_BeforeExit" File="qconsole.qsc"/>
    <QScriptFile Name="QConsole_Exit" File="qconsole.qsc"/>
    <QScriptFile Name="QConsole_MouseDown" File="qconsole.qsc"/>
    <QScriptFile Name="QConsole_KeyDown" File="qconsole.qsc"/>
    <QScriptFile Name="LM_BeforeNavigate2" File="listmanagement.qsc"/>
</QScriptFileDictionary>
```

<QScriptDictionary> ... </QScriptDictionary>

Description

The QScriptDictionary tag begins and ends the list of hooks that will launch IC Scripts. You need to specify a separate `QScriptDictionary` section for the framework as whole as well as for each pane and each control that uses IC Scripts.

Syntax

```
<QScriptDictionary>
    ...IC Script hooks...
</QScriptDictionary>
```

<QScript> ... </QScript>

Description

The QScript tag gives a hook and the associated IC Script.

Syntax

```
<QScriptDictionary>
    <QScript Hook="Hook" [Key="Key"] Name="QScriptName"> </QScript>
</QScriptDictionary>
```

Attribute	Value	Description
Hook	A text string	The name of the hook point at which you want the IC Script to run.
Key (optional)	Any key or a combination using: Alt+<any key> Ctrl+<any key> Shift+<any key>	The key or key combination that runs the script specified in Name. This is an optional parameter that is only used when the IC Scripts are being associated with a pane.

Attribute	Value	Description
QScriptName	A text string	The name of the IC Script you want to run.
Label	Any text string	This is the name that is displayed in the GUI

Example

This example shows the QScript Dictionary for the framework. Examples of pane and control dictionaries can be found in those sections.

```
<QScriptDictionary>
    <!-- Keyboard Accelerator Definitions for Avaya Agent-->
    <QScript Hook="KeyDown" Key="Alt+A" Name="QConsole_KeyDown"/>
    <QScript Hook="KeyDown" Key="Alt+C" Name="QConsole_KeyDown"/>
    <QScript Hook="KeyDown" Key="Alt+D" Name="QConsole_KeyDown"/>
    <QScript Hook="KeyDown" Key="Alt+H" Name="QConsole_KeyDown"/>
    <QScript Hook="KeyDown" Key="Alt+N" Name="QConsole_KeyDown"/>
    <QScript Hook="KeyDown" Key="Alt+O" Name="QConsole_KeyDown"/>
    <QScript Hook="KeyDown" Key="Alt+P" Name="QConsole_KeyDown"/>
    <QScript Hook="KeyDown" Key="Alt+R" Name="QConsole_KeyDown"/>
    <QScript Hook="KeyDown" Key="Alt+S" Name="QConsole_KeyDown"/>
</QScriptDictionary>
```

Specifying frames in your application

The next sections define the frames that will appear in Avaya Agent. You need one QFrame definition within this section for each frame in the application. You can have up to four frames.

<QFrameDictionary> ... </QFrameDictionary>

Description

The QFrameDictionary tag begins and ends the list of frames.

Syntax

```
<QFrameDictionary>
    ...Frame definition statements...
</QFrameDictionary>
```

<QFrame> ... </QFrame>

Description

The QFrame tag begins and ends each frame definition section.

Syntax

```
<QFrameDictionary>
    <QFrame Name="sName" [Width="Width"|Height="Height"]
Orientation="Orientation" Visible="Visible">
.... Pane and control definition statements ....
    </QFrame>
</QFrameDictionary>
```

Attribute	Value	Description
Name	Any text string	The name of the Avaya Agent frame.
Width or Height	A numeric value	If the orientation is Left or Right, use the Width attribute to specify the width in pixels. If the Orientation is Top or Bottom, use the Height attribute to specify the height in pixels.
Orientation	"Left", "Right", "Top", or "Bottom"	Where the frame is anchored on the user's desktop.
Visible	"True" or "False"	Whether the frame is visible. If not, none of the entities it contains will be created and none of its associated IC Scripts will be pushed to the database. Default: True.
Label	Any text string	This is the name that is displayed in the GUI

Example

```
<QFrameDictionary>
    <QFrame Name="Right_Frame" Width="200" Orientation="RIGHT"
    Visible="TRUE">
.... Pane and control definition statements ....
    </QFrame>
</QFrameDictionary>
```

<QStartMenu> ... </QStartMenu>

Description

The QStartMenu tag controls whether there is a Start menu button in the frame.

Syntax

```
<QStartMenu Visible="Visible"> </QStartMenu>
```

Attribute	Value	Description
Visible	“True” or “False”	Whether the Avaya Agent menu is visible or not. Default: True. **Note:** If all frames have the `QStartMenu` tag set to false, or the tag is not defined for any Avaya Agent frame, then the application forces `QStartMenu="True"` for the first visible frame defined in the CDL file.

Example

```
<QStartMenu Visible="True" />
```

<QTab> ... </QTab>

Description

The QTab tag controls what the tabs for Avaya Agent look like.

Syntax

```
<QTab Wrap="Wrap" Appearance="Appearance"> </QTab>
```

Attribute	Value	Description
Wrap	“True” or “False”	Whether the tabs should wrap if they are longer than the pane is wide. If this is set to False and they do not fit on one row, the application adds a scroll bar to the frame. Default: True.
Appearance	"Tabs" or "Buttons"	Whether the tab names look like standard tabs or buttons. Default: Tabs.

Example

```
<QTab Wrap="True" Appearance="Buttons"> </QTab>
```

Specifying panes within a frame

The next section is the Pane Dictionary that collects all the panes that are part of Avaya Agent. For each pane, you specify its name and the IC Scripts that should run at different points during execution. After you specify the IC Scripts, you create a Control Dictionary that lists the OLE controls in the pane. (For more information, see Specifying controls within a pane on page 33.)

<QPaneDictionary> ... </QPaneDictionary>

Description

The QPaneDictionary tag begins and ends the list of panes in the given frame.

Syntax

```
<QPaneDictionary>
    ...Pane definition statements...
</QPaneDictionary>
```

<QPane> ... </QPane>

Description

The QPane tag begins and ends each pane definition section. If you want to keep this pane in the CDL file for reference or future use, set the Visible option to False and it will be ignored when the CDL file is compiled.

Syntax

```
<QPaneDictionary>
    <QPane Name="Name" Label="Label" Visible="Visible">
.... Control definition statements ....
    </QPane>
</QPaneDictionary>
```

Attribute	Value	Description
Name	Any text string	The name of the Avaya Agent pane. This name **must** be unique between all panes in a given frame because IC Script methods use the name to identify the pane. You can, however, have two panes with the same name if they are in different frames.
Label	Any text string	The label that Avaya Agent displays on the pane in the client GUI. If this field is omitted or set to Label="", Avaya Agent uses the Name attribute as the label.
Visible	"True" or "False"	Whether the pane is visible. If not, it will not be created and any associated IC Scripts will not be pushed to the database. Default: True.

Example

```
<QPaneDictionary>
    <QPane Name="Contact" Label="Contact" Visible="TRUE">
.... Control definition statements ....
    </QPane>
</QPaneDictionary>
```

Specifying IC Scripts within a pane

You can specify the IC Scripts within a pane using a QScript Dictionary section exactly the same as the one described for the framework on <QScriptDictionary> ... </QScriptDictionary> on page 27.

The following is an example of this dictionary for a pane:

```
<QPaneDictionary>
    <QPane Name="Media" Visible="True">
<QScriptDictionary>
    <QScript Hook="MouseDown" Name="QConsole_MouseDown" />
    <QScript Hook="KeyDown" Key="Alt+N" Name="QConsole_KeyDown" />
    <QScript Hook="KeyDown" Key="Alt+P" Name="QConsole_KeyDown" />
    <QScript Hook="KeyDown" Key="Alt+D" Name="QConsole_KeyDown" />
    <QScript Hook="KeyDown" Key="Alt+C" Name="QConsole_KeyDown" />
</QScriptDictionary>
... Control definition statements
    </QPane>
</QPaneDictionary>
```

Specifying controls within a pane

The next section is the Control Dictionary. You need one control dictionary section for each pane named above. This section is also where you specify what IC Scripts runs in response to events raised by the control. When you are setting up the control dictionary, you must determine what events might be raised and how you want to handle them. Ensure that the event names you use match exactly with the names that the control uses.

<QControlDictionary> ... </QControlDictionary>

Description

The QControlDictionary tag begins and ends the list of controls in the given pane.

Syntax

```
<QControlDictionary>
    ...Control definition statements...
</QControlDictionary>
```

<QControl> ... </QControl>

Description

The QControl tag begins and ends each control definition section. If you want to keep this control in the CDL file for reference or future use, set the Visible option to False and it will be ignored when the CDL file is compiled.

Syntax

```
<QControlDictionary>
    <QControl Name="Name" ProgId="ProgID" Left="Left" Top="Top"
    Width="Width" Height="Height" Visible="Visible">
.... QScript definition statements ....
    </QControl>
</QControlDictionary>
```

Attribute	Value	Description
Name	*Any text string*	The name of the control. This name *must* be unique no matter what pane or frame the control appears in.
ProgId	*Any text string*	The ProgID of the OLE control. (You can find this name in the documentation for the control.)
Left	*A numeric value, in pixels*	The location of left-hand edge of the control, relative to the left edge of the pane (the X coordinate).
Top	*A numeric value, in pixels*	The amount of space between the top of the pane and the top of the control (the Y coordinate)
Width	*A numeric value, in pixels*	The width of the control in pixels.
Height	*A numeric value, in pixels*	The height of the control in pixels.
Visible	*“True” or “False”*	Whether the control is created. If not, the control is not used and the associated IC Scripts are not pushed to the database.
Label	*Any text string*	This is the name that is displayed in the GUI.

Example

```
<QControlDictionary>
<QControl Name="StatusControl" ProgID="AvayaStatus.StatusCtrl.73" Left="1" Top="1"
Width="193" Height="51" Visible="TRUE"/>
```

```
... QScript Definitions ...
    </QControl>
</QControlDictionary>
```

Specifying IC Scripts within a control

You can specify the IC Scripts within a control using a QScript Dictionary section exactly the same as the one described for the framework on <QScriptDictionary> ... </QScriptDictionary> on page 27. IC Scripts can be attached to application events to override or add to the default action, or to implement business rules.

You can use IC Scripts with Avaya Agent:

- To pop a message when an agent is assigned a phone call or email.
- To determine which requests should get routed to which agents or queues.
- To store information entered by the agent into the Avaya Agent database.

For information on creating, editing, and debugging IC Scripts, as well as information on how to save your IC Scripts to the database, refer to *IC Database Designer Application Reference*.

For information on the available IC Script methods and the event hooks that you can associate with IC Script programs, refer to *IC Scripts Language Reference*.

The following is an example of this dictionary for a control:

```
<QControlDictionary>
  <QControl Name="CallList" ProgID="AvayaSoftPhone.CallListCtrl.73" Left="2" Top="55"
Width="190" Height="43" Visible="TRUE">
    <QScriptDictionary>
      <QScript Event="OnActivate" Name="Shared_OnActivate"/>
      <QScript Event="OnSelect" Name="Shared_OnSelect"/>
    </QScriptDictionary>
  </QControl>
</QControlDictionary>
```

Ending the definition sections

When you are done defining the components in this pane, and then in Avaya Agent, you must end all the definition sections.

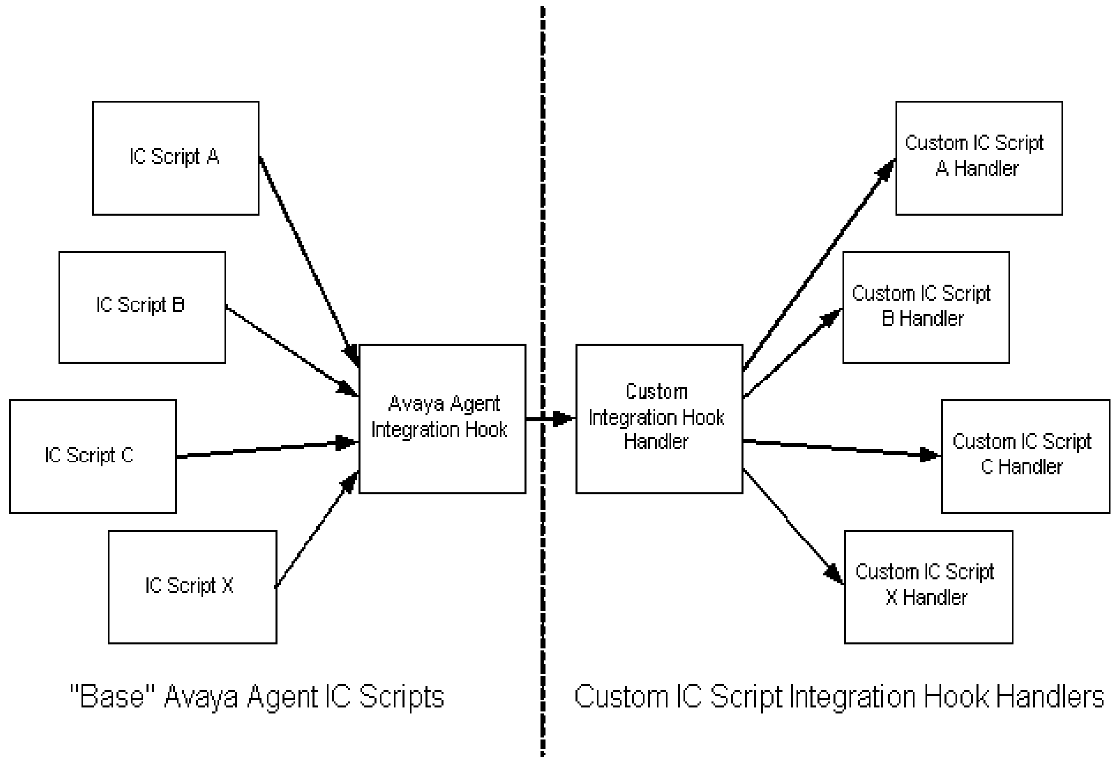
```
</QPane>
</QPaneDictionary>
</QFrame>
</QFrameDictionary>
</QConsole>
```

Customizing Avaya Agent behavior

As discussed in Overview on page 9, Avaya Agent IC Scripts control the behavior of the Avaya Agent. These IC Scripts are the “glue” that binds controls within components as well as components to other components. IC Scripts are programmed to make the Avaya Agent act the way it does. You may need to modify the behavior of the Avaya Agent to suit your business requirements. This chapter describes how to do that. First, you must become familiar with a concept called “Integration Hooks”.

What are integration hooks?

Integration Hooks is the concept of running your custom code from the out-of-the-box IC Scripts. Placed in very key places throughout the out-of-the-box IC Scripts, in very key places, are hooks where a main Integration Hook IC Script is called. The following diagram provides a visual depiction of the concept of Integration Hooks:


IC Script A
IC Script B
IC Script C
IC Script X
Avaya Agent Integration Hook
Custom Integration Hook Handler
Custom IC Script A Handler
Custom IC Script B Handler
Custom IC Script C Handler
Custom IC Script X Handler
"Base" Avaya Agent IC Scripts
Custom IC Script Integration Hook Handlers

In this diagram, calls from IC Scripts are made to the Avaya Agent Integration Hook. Inside the Avaya Agent Integration Hook, the call is added to the Custom Integration Hook Handler. Everything on this half of the diagram is your code. You can write code directly in the Custom Integration Hook Handler or you can call individual Custom Handlers for each of the Hooks you are handling. The latter is the preferred method, because it is easier to maintain as well as to understand what your implementation is composed of.

Information sent to the integration hook

When this Integration Hook IC Script is called, the following parameters are passed with it:

1. **Source** – name of the IC Script from which the main Integration Hook IC Script is being called. Use Source to determine where in Avaya Agent the call to your code is being made.
2. **Source Object** – object that was part of the Source IC Script that is calling the Integration Hook. An example is the TelephonyChangeEvent Object. Most Integration Hooks do not contain a Source Object, however, if they do, you can use this object to perform processing in your Integration Hook Handler.
3. **Source SeqCouple** – parameter used to pass various pieces of data to (and sometimes from) the Integration Hook. Because the Integration Hook is called from many, various places in the Avaya Agent framework, it was determined that a sequence of couples provided the easiest means of transporting this data. The actual names of the couples within the Source SeqCouple vary greatly by Integration Hook.
4. **Cancel Flag** – Integration Hooks exist to give the integrator the capability of changing the behavior of Avaya Agent (within reason). Some Integration Hooks have the capability to completely override the default behavior. Therefore, in order to signify to the calling IC Script, a Cancel Flag is provided. This is a simple Boolean variable, which can be set to Cancel if necessary. Note that for Integration Hooks that DO provide this capability, care should be taken when taking advantage of this.

The following is the syntax of the QConsole_IntegrationHook routine:

```
Sub QConsole_IntegrationHook(sSource As String, iSourceObject As Object,
iSourceSeqCouple As Object, bCancel As Boolean)
```

Start using integration hooks

Integration Hooks are not enabled with Avaya IC. If you must customize Avaya Agent in such a way that requires using them, perform the following steps:

1. In your CDL file, turn on the Integration Hooks:
   a. Open your .cdl file in an editor.
   b. Find the following QProperty Section:

```
<QSection Name="General"
        ScreenPopScript="ScreenPop"
        IntegrationHooksEnabled="False"
        />
```

   c. Set “IntegrationHooksEnabled” = “True”.
   d. Save your .cdl file.
   e. Use Avaya Database Designer to push your .cdl to the database.

2. Create an Integration Hook Handler:
   a. Copy the existing Avaya Agent Integration Hook Handler (QConsole_IntegrationHook.qsc) to a new file (for example: My_IntegrationHookHandler.qsc)
   b. Inside “My_IntegrationHookHandler.qsc”, rename all “QConsole_IntegrationHook” to “My_IntegrationHookHandler”.
   c. Save “My_IntegrationHookHandler.qsc”.
   d. Use Avaya Database Designer to push My_IntegrationHookHandler to the database.
3. Change the Avaya Agent Integration Hook to call your Integration Hook Handler:
   a. Using Avaya Database Designer, locate the “QConsole_IntegrationHook” IC Script.
   b. Add your Integration Hook Handler to the “QScript API Declarations” section.
   c. Read and follow all comments inside “QConsole_IntegrationHook” for very specific directions on where you should place your call to your Integration Hook Handler.
   d. Save “QConsole_IntegrationHook”.
   e. Upload “QConsole_IntegrationHook”.

The above steps bring you to a point where you have enabled the Integration Hooks and your Integration Hook Handler is called every time an Integration Hook is run. Now, you must write all your individual handler IC Scripts that handles each of the Integration Hooks you wish to “plug” into. For each handler you write, you must change your Integration Hook Handler to call the individual handler based on the Source parameter.

Example integration hook code

In order to help jumpstart your customization effort, as well as give a real face to the Integration Hook concept, an entire set of sample Integration Hook code is included with your IC Design installation. This code is located in the following directory: `*IC_INSTALL_DIR*\IC73\design\qconsole\custom`.

In this directory, there is a complete assortment of IC Scripts that demonstrate the use of Integration Hooks from a perspective of doing a full application integration.

To turn on this set of custom code to run in Avaya Agent:

1. First enable the IntegrationHooks in your .cdl file using step 1 from the previous Chapter.
2. Add the `IC_INSTALL_DIR\IC73\design\qconsole\custom` to your ADL Include Path of your application .adl file so that the custom IC Scripts will be loaded into the Database
   a. In Database Designer, open the ccq.adl file.
   b. Add `IC_INSTALL_DIR\IC73\design\qconsole\custom` to the ADL Include Path.
   c. Save the setting.
   d. Select **File** > **Generate Windows Application…** (be sure to check “IC Scripts” and select the "interaction_center" application).
3. Set the “Agent/Desktop.IntegratedApplication” = “Custom” property in IC Manager. Refer to *IC Administration Guide* for instructions.

When you run Avaya Agent, various dialogs are displayed at different points doing startup/shutdown. Right-click on Avaya Agent to see the Context Menu Choices that were added. Everything you see running was accomplished using Integration Hooks without changing any Base IC Scripts.

Important:
Read each of the IC Scripts included in the `IC_INSTALL_DIR\IC73\design\qconsole\custom` directory. They contain many lines of comments that describe everything that is possible in all of the individual Handlers. This set of IC Scripts can be used as a great starting point to get you on your way using Integration Hooks.

Available integration hooks

This section provides a complete list and description of the Integration Hooks that are available within Avaya Agent. Be sure to look at the sample code provided `IC_INSTALL_DIR\IC73design\qconsole\custom` as each individual Integration Hook passes different data in the parameters than what is described in Information sent to the integration hook on page 37.

BlenderClient_ADUChange

This Integration Hook is fired every time an ADU Field that the BlenderClient is monitoring changes. You can add additional fields to the Blender Clients monitor criteria. See the sample code in `IC_INSTALL_DIR\IC73\design\qconsole\custom` for an example of this.

BlenderClient_AgentStateEvent

This Integration Hook is fired every time the Agent's state changes. This allows you to do custom coding for when an agent moves from one state to another.

BlenderClient_ChannelStateEvent

This Integration Hook is fired every time the Agent's Channel state changes. This allows you to do custom coding for when an agent’s channel moves from one state to another.

CHBrowser_Initialize

This Integration Hook is fired when the CHBrowser is being Initialized. This is a special place where you can perform additional configuration on the CHBrowser. If you choose to, you can completely reconfigure the CHBrowser, then set the Cancel flag of the routine to "True" so the out-of-the-box initialization is not executed.

CHBrowser_OnContactSelected

This Integration Hook is fired when an item in the CHBrowser is selected. There are 3 things you can do with this Integration Hook:

1. Nothing, use the default behavior.
2. Using the Key, do the work to determine how to retrieve the mediainteraction record, and set the appropriate QBEKey and QBEValue into the sMediaInteractionQBEKey and the sMediaInteractionQBEValue fields.
3. Customize to your specific requirements. You can do that by just doing what you want here and setting Cancel flag to "True".

MailEngine_TaskDeclined

This Integration Hook is fired from the MailEngine when an incoming email is rejected. This is used to tell if RONA occurred and/or if the agent manually rejected an email.

> **Note:**
> The EDUID parameter is not always filled. Many times this event is raised before the contact was actually delivered to Avaya Agent.

MailEngine_TransferResponse

This Integration Hook is fired from the MailEngine in response to a Transfer request. It is used to determine if the Transfer request succeeded or failed and why.

PhoneEngine_OnCoreStateChanged

This Integration Hook is fired every time an EDU Field changes that the Phone Engine is monitoring. You can add other fields to the Phone Engine's list of monitored fields. See the sample code for an example of this.

PhoneEngine_OnTelephonyStateChanged

This Integration Hook is fired whenever a change occurs in the Avaya Agent Softphone. This is a full event source of all the Telephony Events you might need to use in a customization.

QConsole_AddContact

This Integration Hook is fired for every contact that enters Avaya Agent. Here you would do whatever needs to happen to your custom component when a contact is added to Avaya Agent.

QConsole_BuildActiveContactCriteria

This Integration Hook is fired during the QConsole_AddContact process. It is used for building the criteria that will be used to fill the Active Contact Viewer with contacts related to the one being handled. This Integration Hook and the one for QConsole_BuildCHBrowserInfo go hand-in-hand. If you are doing something custom to retrieve historical records in the CHBrowser, then you would probably do something custom to fill the Contacts in the Active Contact Viewer. Like the choices found in QConsole_BuildCHBrowserInfo, the following are options for what you can do with this Integration Hook.

1. Nothing, use the default behavior.
2. Fill the Criteria parameter with the criteria for finding the Active Contacts. Your criteria is mutually exclusive and an out-of-the-box criteria is not used (except for the vdu_id).
3. Customize to your specific requirements. You can do that by doing what you want here, set the Cancel flag to "True", then fill the ActiveContactCriteria variable with the *exact* VDU.Find criteria you want.

QConsole_BuildCHBrowserInfo

This Integration Hook is fired during the QConsole_AddContact process. It is used for building the information that will go into the process of filling the CHBrowser with records. The following are options for what you can do with this Integration Hook.

1. Nothing, use the default behavior.
2. Fill the CHBrowserField parameter with a field you want to use for backfilling records. In the CHBrowserValue parameter, place the value that will be used to search in the CHBrowserField. If you don't have a value to put into CHBrowserValue parameter, leave the field blank and fill the DummyCHBrowserValue parameter with a value that will cause the CHBrowser to retrieve no records. This is only be used if nothing was found in the out-of-the-box code either.
3. Customize to your specific requirements. You can do that by just doing what you want here and setting Cancel flag to "True".

QConsole_CompleteContact

This Integration Hook is fired for a contact to complete it. This is after QConsole_WrapContact and calls QConsole_RemoveContact. You perform any Contact related cleanup to your custom component.

QConsole_ExitComponents

This Integration Hook is fired during the shutdown of Avaya Agent. Basically, during shutdown, you log out of your custom component, then, if this succeeds, invoke additional methods to completely shut your component down.

QConsole_InitializeComponents

This Integration Hook is fired during the startup of Avaya Agent. It should be used for doing things like setting properties and running methods on controls in your integration that are done once, at the beginning. Remember that anything you do here will need to be "undone" in the QConsole_Exit Integration Hook.

Another thing that should be done here is the setting of properties on other controls within Avaya Agent. Because the Integration Hook that is fired for this routine is last, you can change any properties that were set by the Components individual Initialize routine. Remember that wherever possible, try to use configuration either in the Agent Properties and/or in the .cdl settings where available first.

QConsole_LoginComponents

This Integration Hook is fired during the startup of Avaya Agent, but after all controls are initialized. It should be used for doing things to your component (or other components) that can only be done after they have been logged into.

QConsole_MouseDown

This Integration Hook is fired whenever someone right-clicks on Avaya Agent. This should be used to add additional Context Menu options to Avaya Agent. These menu items can be added in any order as Avaya Agent automatically sorts in the following way: None Component Items Alphabetically, followed by Component names Alphabetically with sub menu items being sorted within component alphabetically as well.

QConsole_PerformScreenPopFromEDU

This Integration Hook is fired when a Contact has met the configured rule for being popped. When using Integration Hooks, the standard, configured ScreenPop IC Script is no longer run. It is up to you to do something here.

QConsole_RemoveContact

This Integration Hook is fired when a contact is physically being removed from Avaya Agent. This is typically called from QConsole_CompleteContact. If your custom component has some relationship to the contact in Avaya Agent, you should update it here.

QConsole_ShowContact

This Integration Hook is fired when a contact is being shown in Avaya Agent. If your custom component needs to be kept in synch with the currently shown contact in Avaya Agent, you could add code to do this here.

QConsole_WrapContact

This Integration Hook is fired for every contact when it is entering the wrapped state. Note that this event handler will only be fired if the wrap-up state has been entered. If WrapUp is not enabled, or if the requirements for entering wrap-up are not met, this Integration Hook is not fired.

Shared_OnActivate

This Integration Hook is fired for controls that expose the OnActivate event. Those controls are the Chat List and Call List.

Shared_OnSelect

This Integration Hook is fired for controls that expose the OnSelect event. Those controls are the Chat List and Call List.

Softphone_PlaceCall

This Integration Hook is fired whenever a call is placed...just before placing the call. You have two options:

1. If you modify the sDestination variable, this is used when the call is placed. This is where you could prepend Feature Access Codes (such as 9).
2. Since you have the EDUID, you can place information into the EDU before the call is placed. This lets you prepopulate the EDU with something that is needed when the call is received somewhere.

TaskList_Change

This Integration Hook is fired whenever there is a change in the Avaya Agent TaskList. This is a full event source of all the Email Events you might need to use in a customization.

TaskList_GuiActivate

This Integration Hook is fired when there is a GUI Activation of a task in the TaskList. Gui Activation differs from Activation, as Active can be a state of a test. Gui Activate come on tasks that are double-clicked or [Enter] hit on, regardless of their state.

UAD_CancelContact

This Integration Hook is fired when a contact is being cancelled from the UAD. Sometimes you might need to do something in your own integration, like update an internal state, as a result of this.

UAD_Initialize

This Integration Hook is fired when the UAD is being Initialized. This is a special place where you can do extra configuration on the UAD.

UAD_Login

This Integration Hook is fired during login for the UAD. This is a special place where you can do extra configuration on the UAD that must be done after Login. This can be used to add additional Contact Lists to the UAD based on Workgroup names. Note that when loading Contact lists, the UAD must be first loaded. Usually, the UAD loads on the first showing (which is why it takes a few seconds to show the very first time). If you do not mind taking the hit at login, you may do the UAD.Load here (as well as add the Contact Lists). If loading at login is not an option then you can use the UAD_Show Integration Hook instead.

UAD_Show

This Integration Hook is fired just before the UAD is displayed. You can use the Cancel flag to prevent the UAD from being displayed. Also, as explained in the UAD_Login Integration Hook description, you might want to load contact lists only at the time the UAD is first displayed.

UAD_UADStateChange

This Integration Hook is fired whenever something happens in the UAD. You can use this Integration Hook for taking action before the Avaya Agent handlers run. This means you can do things like manipulate the EDU or other things.

WACEngine_AgentStateChanged

This Integration Hook is fired from the Web Agent through the WACEngine as its various servers go through various state changes.

WACEngine_ErrorOccurred

This Integration Hook is fired from the Web Agent through the WACEngine when certain errors are encountered.

WACEngine_PerformConsoleAction

This Integration Hook is fired from the Web Agent through the WACEngine when it wants to request an action be done in Avaya Agent. Examples of this is for Supervisor monitoring ("display_edu"/"remove_edu")...forcing windows to come to the top ("show_window")...and launching URLs (display_url). The action you will most likely find a need to handle is "shellexecute". This is sent when the Web Agent wants to open an attachment for an email. In your implementation, you might want to launch a custom app, or, as we have found with other file types, fix the launch to work properly.

WebEngine_ActivateCallback

This Integration Hook is fired whenever a callback is placed for a chat...just before placing the call. The number is coming from the Web Agent GUI. This will (unless the agent changed) contain the original spaces in the phonenumber as arranged on the website. You can use this to your advantage as the PhoneNumber variable can be modified. Also, this is where you could potentially pre-pend Feature Access Codes (such as 9), although, Softphone_PlaceCall is used to place the call, so that Integration Hook will be hit as well.

WebEngine_TaskDeclined

This Integration Hook is fired from the WebEngine when an arriving chat is rejected. This is used to tell if RONA occurred and/or if the agent manually rejected a chat. Note that the EDUID parameter is not always filled. Many times this event is raised before the contact was actually delivered to Avaya Agent.

WebEngine_TransferResponse

This Integration Hook is fired from the WebEngine in response to a Conference request. It is used to determine if it succeeded or failed and why.

WebEngine_WebStateChanged

This Integration Hook is fired whenever there is a change in the Avaya Agent WebEngine. This is a full event source of all the Chat Events you might need to use in a customization.

Connecting to an external database

When you deploy Avaya IC, you may need to link records from a third party application to record in the IC Repository. You can create the required links with the mapping tables (ex*map tables) that are provided with IC Repository.

In standard Avaya IC deployments, the following records are typically linked to third party application records:

- Customers
- Organizations
- Contacts

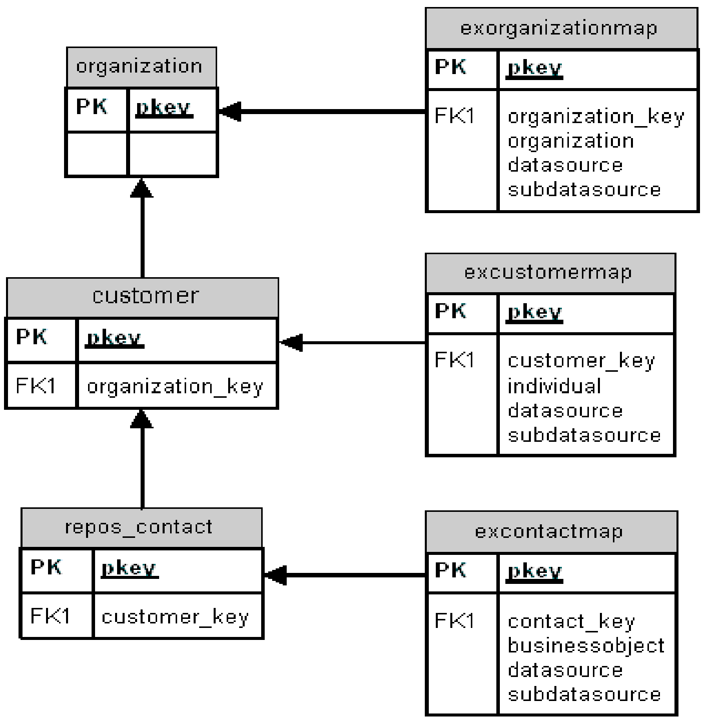
If you build a relationship between the third party application records and these Avaya IC records, you can perform reporting that links IC Repository and the third party application database. For example, if you relate a Customer record in the third party application database with a Customer record in IC Repository, you can easily perform database lookups to retrieve Customer information from the third party application database.

Ex*map tables

The ex*map tables are the mapping tables in IC Repository These tables store the relationships between entities in IC Repository and third party application databases.

Structure and relationships of ex*map tables

The following diagram shows the structures/relationships of the mapping tables:



List of ex*map tables

The following list defines the ex*map tables:

excontactmap - Used to map records in a third party application to the contact's record in Avaya IC, which allows for an association between an Avaya IC contact and the business objects associated with that contact in the external system. For example, you can link an order, task, or other object that was used or created as a result of the contact. The object can then be used to provide a link back into the 3rd party application for historical and reporting purposes.

excustomermap - Used to map records in a third party application to the customer record in Avaya IC. This allows for an association to the customer data without redirecting the customer table. The goal is to provide Avaya IC with the capability to store customer data for Avaya specific data.

exorganizationmap - Used to map records in a third party application to the organization record in Avaya IC. This allows for an association to the organization data without redirecting the organization table. The goal is to provide Avaya IC with the capability to store organization data for Avaya specific data.

Each of these ex*map tables have two fields in common, the datasource column and the subdatasource column. The data sources are used to be able to re-access the data linked to the table. By looking at these fields, a program can determine how to fetch the data when necessary to retrieve.

Columns in ex*map tables

The following list defines the columns in the ex*map tables:

datasource - The source for the system where the data is located. For example, if you integrate with a system called “abc”, you would set “datasource” to “abc” when you create records in each of the ex*map tables. This is a text field, so you can set this value to anything you need.

subdatasource - The object within the datasource to which you are relating. If you are linking to a record from the “Bar” table in your "abc" system, you would set “subdatasource” to “Bar”. This is a text field, so you can set this value to anything you need.

individual/organization/businessobject - The foreign key fields of the ex*map tables. Put the unique key from the datasource.subdatasource record you want to relate to the record in these fields. This is a text field, so you can set this value to anything you need.

Populating the Ex*map Tables

Each of the ex*map tables are populated differently. The following table provides a list of which methods can be used to populate each of the ex*map tables. These methods are defined in the section immediately after this table.

Table name	**Methods used to populated**
excontactmap	Direct access via SQL IC Script Workflow EDU (via Report Server)
excustomermap	Direct access via SQL IC Script Workflow
exorganizationmap	Direct access via SQL IC Script Workflow

IC Script or workflow

You can use an IC Script or a workflow to populate the ex*map tables. An IC Script or workflow can populate the table when a contact is routed or qualified, when authentication is performed, or when an agent uses Avaya Agent.

Direct access to database via SQL

You can use SQL and the tools available with your database to directly load records into the tables. This method can be useful if you need to maintain a constant relationship between Customer and Organization data in a third party application database synchronized with IC Repository.

EDU (via Report Server)

For the excontactmap table only, you can use an EDU to populate this table. When an EDU retires, the Report Server takes the EDU and runs it through mapping rules to create and store records in IC Repository. Since the excontactmap table is related to the IC Repository records, you can use the out-of-the-box mapping rules to create excontactmap records.

IC Repository must have the proper structure to use this method. The required structure is:

```
excontactmap.<n>.businessobject
excontactmap.<n>.datasource
excontactmap.<n>.subdatasource
```

An example of this would be for the “abc” system, “Bar” table, “xyz” example. To create a record in the excontactmap when the EDU retired, you must write:

```
excontactmap.+.businessobject
excontactmap.!.datasource
excontactmap.!.subdatasource
```

`which, if the first excontactmap container written, is written as:`

```
excontactmap.1.businessobject
excontactmap.1.datasource
excontactmap.1.subdatasource
```

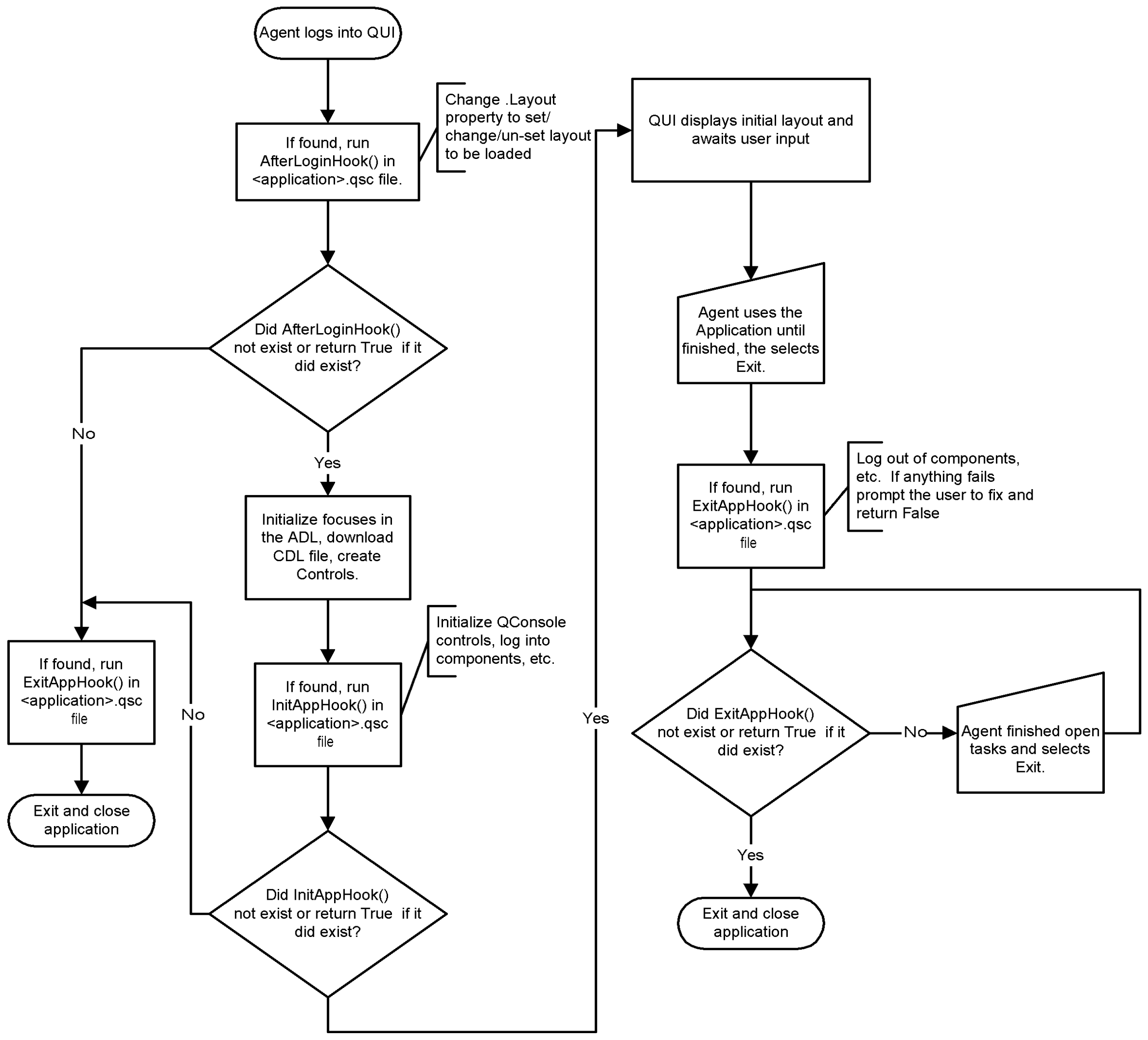
Chapter 5: Initialization and exit hooks

This chapter discusses the Avaya Agent Initialization and Exit Hooks that can be used for the initialization and exit processing for your customizations.

This section includes the following topics:

Avaya Agent initialization and exit hooks

The following flowchart shows the process when an agent logs into, and eventually out of, Avaya Agent. From this flowchart, you can tell when the main IC Scripts hooks are run.


Agent logs into QUI
If found, run AfterLoginHook() in <application>.qsc file.
Change .Layout property to set/ change/un-set layout to be loaded
Did AfterLoginHook() not exist or return True if it did exist?
No
Yes
Initialize focuses in the ADL, download CDL file, create Controls.
If found, run InitAppHook() in <application>.qsc file
Initialize QConsole controls, log into components, etc.
Did InitAppHook() not exist or return True if it did exist?
No
Yes
If found, run ExitAppHook() in <application>.qsc file
Exit and close application
QUI displays initial layout and awaits user input
Agent uses the Application until finished, the selects Exit.
If found, run ExitAppHook() in <application>.qsc file
Log out of components, etc. If anything fails prompt the user to fix and return False
Did ExitAppHook() not exist or return True if it did exist?
No
Agent finished open tasks and selects Exit.
Yes
Exit and close application

After the agent has successfully logged into the database and before Avaya Agent downloads the CDL layout file specified on the command line, Avaya Agent tries to execute the IC Script `AfterLoginHook`. If this IC Script exists, Avaya Agent waits for the IC Script to run. Out-of-the-box, this IC Script is used to read the Agent/Desktop/Layout property value, and, if needed, set into the `SApplication.Layout` property. When you change this property, it overrides the CDL file specified on the command line based on whatever parameters you choose. For details about adding layouts to the database, refer to *IC Database Designer Application Reference*.

After the determination of which layout to use, Avaya Agent compares the database layout's modified time with the layout file cached on the agent's machine. If the database version is newer, the new layout is retrieved and the agent is required to re-log in to any components specified in the Login section of the new CDL.

Note:

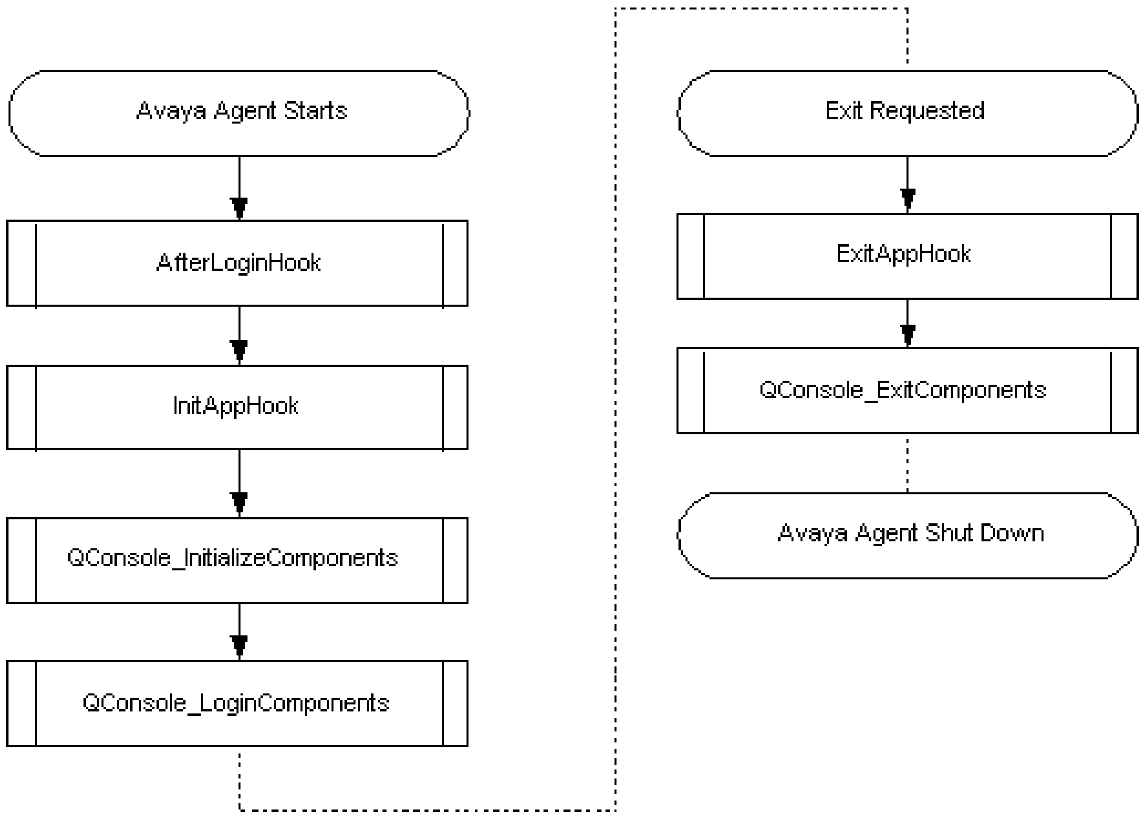
It does not matter where the changes actually are in the CDL file—the agent must log in every time a new CDL file is retrieved from the database.

When the agent has successfully logged in, Avaya Agent proceeds to initialize the applications. If found, it runs the `InitAppHook` IC Script in the `<application>.qsc` file before it initializes and displays the Avaya Agent controls and logs into the other components specified in the CDL.

Finally, when the agent selects Exit, Avaya Agent runs the `ExitAppHook` IC Script in the `<application>.qsc` file. You can use this IC Script to ensure that agents have wrapped up all open contacts before they can exit the application.

For details about writing IC Scripts and a complete list of the events emitted by the Avaya Interaction Center, refer to *IC Scripts Language Reference*. For a complete listing of IC Scripts used by Avaya Agent, and general CDL settings, see IC Scripts on page 111.

The following diagram illustrates the IC Scripts that are involved during the initialization and exit process:



Relevant integration hooks

The following integration hooks are directly related to the Avaya Agent Initialization and Exit process. You can use these hooks to enhance or modify the behavior of the Avaya Agent by creating Integration Hook Handlers as described in Customizing Avaya Agent behavior on page 36.

- QConsole_InitializeComponents
- QConsole_LoginComponents
- QConsole_ExitComponents

Chapter 6: Contact handling

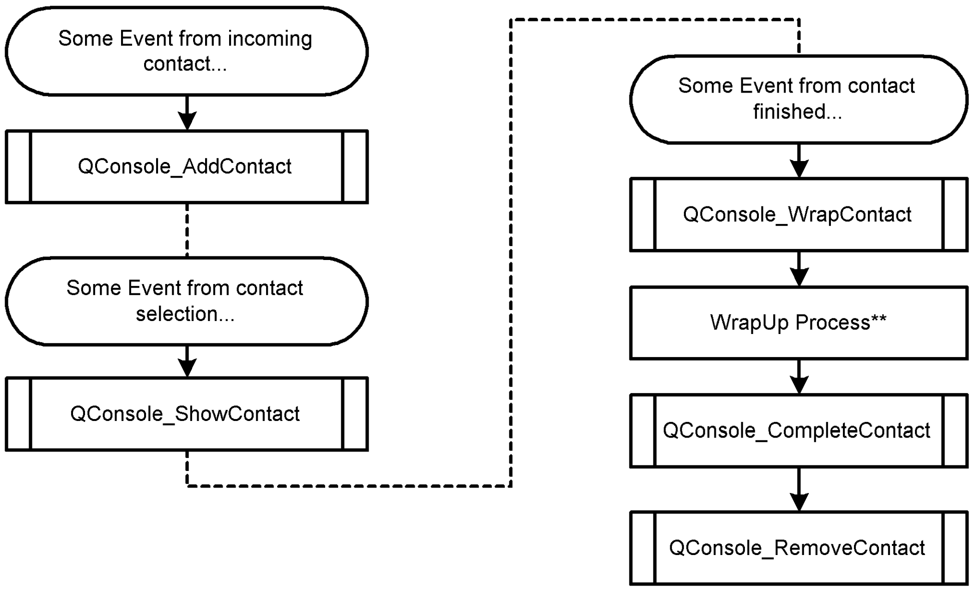
As described in Chapter 2: Overview, the out-of-the-box Avaya Agent layout has separate components for handling different types of media contacts. When they are integrated into the design, however, Avaya Agent treats all contacts uniformly so you can set up a single strategy that can be applied to any and all media components.

This chapter describes the way contacts are handled, and discusses the Avaya Agent IC Scripts that you could use to integrate a new media channel if necessary.

This section includes the following topics:

Lifecycle of a contact

The following diagram outlines the lifecycle of a contact within Avaya Agent:



** For more information, see Chapter 11: Contact wrapup on page 103.

The previous diagram shows that any media channel can be integrated with Avaya Agent if it emits events when a contact:

- comes in.
- is selected.
- is finished.

Because all contacts are handled through IC Scripts, changing the contact handling behavior can be done easily and uniformly.

Avaya Agent contact handling IC Scripts

In this section we will take a look at the IC Scripts that are used for handling contacts in Avaya Agent. The first of these IC Scripts is QConsole_AddContact.

QConsole_AddContact is run from a Event Handler of an incoming contact. It performs the following functions:

1. It activates the Avaya Agent Contact tab so information about the incoming contact can be displayed.
2. It adds the information needed for handling the contact to the contact's EDU. This information includes things like media type, screen pop values, customer key information, and contact labeling.
3. It adds the contact to an internal store of contacts via ContactList_Add. This will be used later for accessing information about all contacts in Avaya Agent.
4. If there is only one contact in Avaya Agent and Screen Pop is enabled, it performs a screen pop. (For details, see Screen pop on page 58.)
5. It adds the contact to the EDU Viewer. (For details, see EDU Viewer on page 70.)
6. Finally, it adds the contact to the Contact History Browser.

Multiple contacts can be added to Avaya Agent using QConsole_AddContact. While the contact is still active in Avaya Agent, the information about the contact remains in the Contact Tab.

You can use QConsole_ShowContact to select a contact within one media, or to change to a contact in a different media. QConsole_ShowContact automatically refreshes any components displaying information about the newly-selected contact. In the out-of-the-box layout, this includes both the EDU Viewer and Contact History Browser.

To close, or wrap up, a contact, you can use the QConsole_WrapContact IC Script. Out-of-the-box, there are two different means of performing Contact WrapUp; through the WrapUp Dialog or Prompter. (For details, see Chapter 11: Contact wrapup). When the WrapUp Process is completed, then the QConsole_CompleteContact IC Script performs any logic necessary to “finish-up” the contact. At the end of QConsole_CompleteContact, QConsole_RemoveContact removes information about the contact from the components in Avaya Agent, including the EDU Viewer and Contact History Browser.

Screen pop

This section provides an overview of how information about contacts coming into Avaya Agent is shown in the application. An application can retrieve relevant customer information for display on an agent's desktop just as a customer's contact arrives. This display of customer information is usually referred to as a **screen pop**.

As described above, when a contact comes into Avaya Agent, QConsole_AddContact is run. In this IC Script, based on information in the EDU and type of contact, more information is put into the EDU so that a screen pop can be performed at any time. If the contact is associated with a known customer, then this information is the customer record's key. If not, then it is the identifier for the customer specific to the media the contact came in on.

Avaya Agent uses the IC Script QConsole_PerformsScreenPopFromEDU when a screen pop is done. This IC Script uses information put into the EDU by QConsole_AddContact, then runs the ScreenPop IC Script.

You can customize the Screen Pop behavior in either of the following ways:

- Invent a new IC Script that will do the Screen Pop to your specifications. Set the name into the "General/ScreenPopScript" QProperty in the .cdl. Note that to do this, your syntax of the Screen Pop Script you define must match that of the ScreenPop IC Script found in <application>.qsc.
- Hook into the QConsole_PerformScreenPopFromEDU Integration Hook.

Note:
When you have enabled Integration Hooks, Option 1 above no longer applies. When you are using Integration Hooks (for other things), one side effect is that you MUST handle the QConsole_PerformScreenPopFromEDU Integration Hook yourself.

Unified Agent Directory

The Unified Agent Directory (UAD) is used for initiating new contacts and handling existing contacts though transfers, consultative conferences, or conferences. IC Scripts surround the initialization, showing, hiding, and operating of the UAD. Therefore, if you want to implement custom behavior surrounding the UAD, you can do so by taking advantage of the various Integration Hooks that are fired from these IC Scripts. You may want to do things like add Contact Lists, write extra data to the EDU, etc. All of these things can be accomplished using the Integration Hook.

Unified Agent Directory API

This section describes the Unified Agent Directory (UAD) API.

SetSite (method)

Description

This method sets the default site in the site drop down of the UAD to the value passed as parameter.

Syntax

```
Boolean StartWrapUp(siteName As String)
```

Value	Description
siteName	Name of the site which should be selected by default when the UAD opens. The value is case sensitive and should exactly match with the site name configured using the IC manager.

Returns

The method returns a Boolean. True is returned if AARC was able to set the site, else false is returned and default site for the agent is set.

In case the value of the default site to be selected in the site drop down has to be changed frequently, a corresponding property can be declared in the IC manager and its value can be used in the scripts. The Administrator will then not be required to redeploy the scripts each time the value is changed.

For example, 'SiteSelection' property of Datatype string can be created in 'Avaya\Desktop\Directory'.

SetCustomTabFocus (method)

Description

This method sets the default focus on the tab whose value is passed as a parameter. The method returns true if the argument is valid, else false is returned and default Tab is set for the agent.

Syntax

```
Boolean SetCustomTabFocus (TabSel As Long)
```

Value	Description
TabSel	The corresponding long value of the tab on which the focus should be set. For instance in OOTB AARC the value of Queues tab is 0 and Agents tab is 1.

Returns

The method returns a Boolean. True is returned if the argument is valid, else false is returned and default Tab is set for the agent.

In case the value of the Tab to be focused has to be frequently changed, a corresponding property can be declared in the IC manager and its value can be used in the scripts. The Administrator will then not be required to redeploy the scripts each time the value is changed.

For example, 'TabFocus' property of Datatype Integer can be created in 'Avaya\Desktop\Directory'.

CDL settings

The following CDL settings are applicable to Contact Handling:

QSection Name	Attribute(s)	Description
General	ScreenPopScript	the name of the IC Script that is used for screen popping.

Relevant integration hooks

The following Integration Hooks are directly related to Contact Handling. Use them to either enhance or change the behavior by creating Integration Hook Handlers as described in Customizing Avaya Agent behavior on page 36.

- QConsole_AddContact
- QConsole_CompleteContact
- QConsole_PerformScreenPopFromEDU

- QConsole_RemoveContact
- QConsole_ShowContact
- QConsole_WrapContact
- UAD_CancelContact
- UAD_Initialize
- UAD_Login
- UAD_Show
- UAD_UADStateChange

Chapter 7: Core services

The Core control, the Status control, and the Blender Client control make up the core services for Avaya Agent. This chapter discusses each of these controls in detail.

This section includes the following topics:

Core control

The Core control is the pivotal point of all contact handling within Avaya Agent. The main responsibility of Core is to provide EDU access to all of the Media Engines and IC Scripts in Avaya Agent. This control is part of the Phone Engine.

Status control

The Status control is used to change the status of multimedia channels and shows a visual indication of that status in Avaya Agent. It interacts with the invisible QBlenderClient control, which performs the majority of the processing, and QCore services. Optionally, it may interact with the Async control to provide status information of a long-running operation external to Avaya Agent. Such an operation may be an OLE Automation function, for example to perform a screen pop into an agent desktop application.

Status states

The Status Control can be in any of the following states:

State	Description
available	The Agent is available to be assigned tasks in one or more channels.
auxwork	The Agent is not available to work on any tasks in any channel. (This is denoted by red Xs over the Agent and channel icons.)
init auxwork	The Agent has requested a transition from Available to Unavailable, but there are still open tasks that must be wrapped up before the transition can be completed. (This is denoted by a grey X over the Agent icon.)

Similarly, each channel control can be in one of these states:

State	Description
available	The channel is open and the agent can be assigned tasks of this type.
busy	The channel is closed and no new tasks can be assigned. (This is denoted by a red X over the icon.)
logged out	The channel is unavailable. (This is denoted by a grey X over the icon.)

Status modes

There are two modes of operation in the Status Control, automatic and manual. In automatic mode, the maximum number of allowable tasks/channel is set by the system, and any change the agent makes to his or her availability applies to all channels.

In manual mode, the agent can set their own load for each channel, and they can set Availability or Unavailability separately for each channel.

Note:
The maximum number of allowable tasks is controlled by Blender Flows. For more information, see *Avaya Workflow Designer User Guide*, “Blender Flows”. You can prevent the agent from changing his load level, on a media by media basis. To do so, set a variable in the agent’s Agent Data Unit (ADU), media.privileges, via a Workflow. Set to "true" to enable manual mode or set to "false" (or omit the setting) to disable manual mode. For more information on ADUs, see *Agent Data Unit Server Programmer Guide*.

Automatic mode

In automatic mode, only the Agent icon looks like a button because changes affect all channels. The agent can toggle between Available and Unavailable by clicking the Agent icon. If the Agent attempts to designate themselves as Unavailable while they have an active task, Avaya Agent goes into the init auxwork state and displays a message requesting that the agent wrap up any active tasks. When the last active task is closed, Avaya Agent goes into the auxwork state and the agent becomes Unavailable.

At this point, Avaya Agent displays a red X over the agent icon and each of the channel icons to indicate the agent is unavailable.

Manual mode

In manual mode, the agent may transition each of the media channels manually from available to busy, or from busy to available. To indicate this, Avaya Agent displays the channel icons as buttons.

In addition, the agent may change the channel load values that control how many tasks the agent may have at any one time on a given media channel. Changing these load values may only be performed in manual mode.

To go into manual mode, the agent:

1. Right-clicks Agent icon.
2. Selects Options from the pop-up menu.

   Avaya Agent displays the Channel Load dialog box.
3. Selects the Manual Mode check box.

   Avaya Agent changes to manual mode and allows the agent to change the channel loads

Retrieving and setting EDU data

For each contact received by an agent there is an accompanying EDU that contains information throughout the lifetime of that contact. Among the information that can be contained within the EDU is the customer specific information such as Account Number or Service ID. EDU data is accessible as name/value attribute pairs. To retrieve a value, methods exist to specify the attribute name (e.g., "account"). The value is returned in a string representation. In addition, methods exist that allow name/value attributes to be set in the EDU.

Blender client control

The Blender Client control is the control that runs media blending within Avaya Agent. When logging into CoreServices, the BlenderClient assigns to the Blender server and ADU server. It processes ADU change events in order to set Agent availability. It also is advised of all Media channel engines in Avaya Agent so that it may set availability.

Note:
For a full description of Blender processing, refer to Blender chapter in the *Avaya Workflow Designer User Guide*.

CDL settings

To use the Core services, set the following CDL parameters:

QSection Name	Attribute(s)	Description
CoreServices	MakeChannelBusyWaitTime	amount of time to wait (milliseconds) for the result of the IC Script CoreServices_MakeChannelBusy.
Core Services	ServerRestartRetryCount	number of times to retry to restart a failed server.
Core Services	ServerRestartRetryWaitTime	number of seconds to wait between each try to restart a failed server.
CoreServices	ContextMenuKey	the ASCII key combination that displays the Context menu. This must be pressed when the Status Control has focus.

Relevant integration hooks

The following Integration Hooks are directly related to Core Services. Use them to either enhance or change the behavior by creating Integration Hook Handlers as described in Customizing Avaya Agent behavior on page 36.

- BlenderClient_ADUChange
- BlenderClient_AgentStateEvent

- BlenderClient_ChannelStateEvent
- PhoneEngine_OnCoreStateChange

Chapter 8: Contact viewing

There are three controls that show information about the contact. The Active Contact Viewer shows any active contacts related to the customer of the current contact (such as emails in the system). The EDU Viewer shows information from the current contact's EDU. And finally, the Contact History Browser shows all of the previous contacts made by the customer of the current contact.

This section includes the following topics:

Active Contact Viewer

The Active Contact Viewer is a control that lets you look at any arbitrary set of existing EDUs using a given set of criteria, and it is represented by an icon that resides on the GUI. When criteria for EDU is given and found, the icon changes allowing you to click and bring up a window containing an EDU Viewer displaying all EDUs matching the criteria you specified.

The Active Contact Viewer has been integrated with Avaya Agent's Contact History Browser in order to display the active contacts related to the currently selected customer. Avaya IC stores the EDU criteria for each contact in the Tag property of the Tabs in the ContactHistoryBrowser. When a tab is selected in the Contact History Browser, the criteria is pulled from that Tab and applied to the Active Contact Viewer.

CDL settings

To use the Active Contact Viewer, you need to set the following CDL parameters:

QSection Name	Attribute(s)	Description
ACViewer	TabLabelEDUField	Specifies which EDU field to use for the label of the Tabs of the EDU Viewer in the Active Contact Viewer.
ACViewer	EDUFindScope	Specifies the scope to use for finding related EDU's. This matches the possible values for EDU Monitoring.

Relevant integration hooks

The following Integration Hook is directly related to the Active Contact Viewer. Use them to either enhance or change the behavior by creating Integration Hook Handlers as described in Customizing Avaya Agent behavior on page 36.

- QConsole_BuildActiveContactCriteria

EDU Viewer

For all incoming or outgoing contacts such as a telephone call or an email message, Avaya IC creates an EDU. The EDU consists of a sequence of name/value pairs called couples that contain information relating to the contact.

A set of couples is called a sequence of couples. For example, in an environment with telephony integration, each telephone call arriving at or originating on the PBX triggers the creation of a permanent call-detail record stored in the EDU.

For example, a telephony-specific EDU may contain:

- The time the call arrived at the PBX.
- VRU information entered by the caller.
- Transfers between agents.
- The time the call concluded.
- The customer service actions performed by the agents.

In addition to uniquely identifying each contact, the EDU collects information about activity that is performed on behalf of the contact and updates that information as the contact traverses the contact center.

The EDU Viewer is an ActiveX control that displays the information contained in the EDU. When a contact is routed to an agent's desktop, the EDU Viewer looks at the stylesheet name/value couple in the EDU and retrieves the name of the associated Extensible Style Language (XSL) stylesheet template. If it finds that stylesheet in the database, it uses it to display the information in the EDU. Otherwise, it uses the default stylesheet.

If additional contacts come in while the agent is still viewing the first one, Avaya Agent creates a tab for each contact so the agent can preview it before he or she actually accepts communication with that contact.

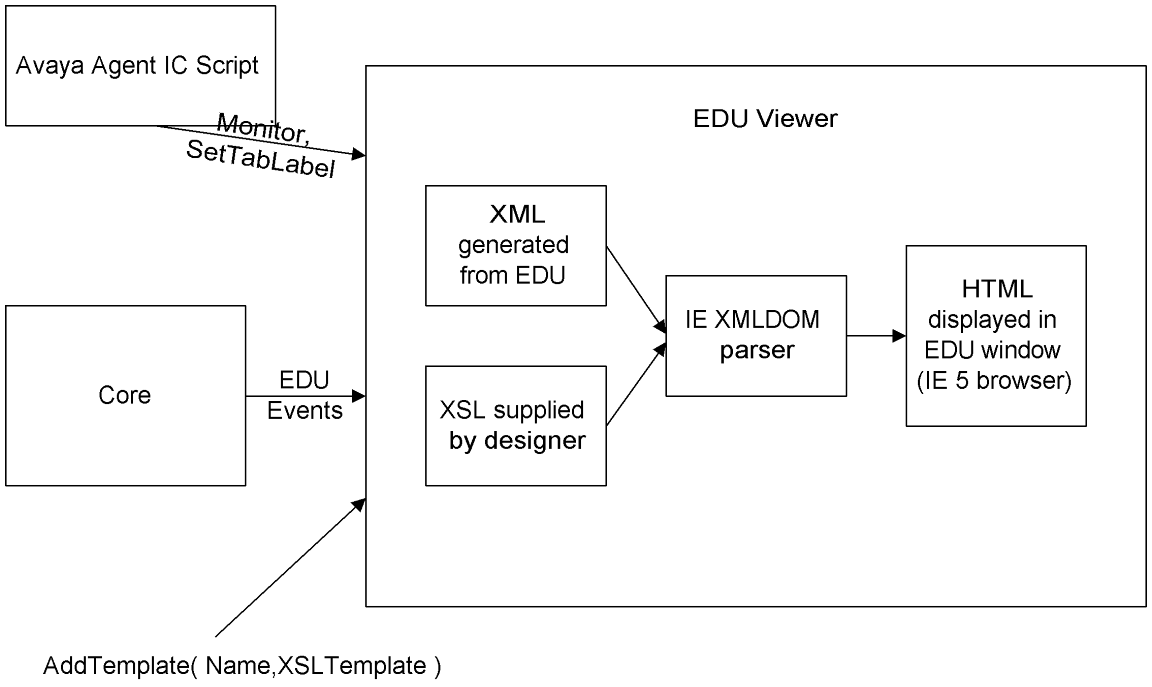
As the system integrator, you can use the default stylesheet for all calls, emails, and chats that will be displayed in the EDU Viewer, or you can create a unique XSL stylesheet for each type. If you want to have different stylesheets, then you must also ensure your EDU contains the name/value couple where name is "stylesheet" and value is a string that uniquely identifies that stylesheet.

If you have Routing Engine, you can add the stylesheet name/value couple to the EDU as part of the call flow. Otherwise, you can add the pair by invoking the VDU.Set method on the VDU server. For details on using Routing Engine, refer to *Avaya Workflow Designer User Guide*. For details on the VDI.Set method, refer to *Electronic Data Unit Server Programmer Guide*.

You may also want to write IC Scripts that are run when various events are triggered in the EDU Viewer. For an in-depth discussion of how to write IC Scripts, as well as a list of the general methods that are available, refer to *IC Scripts Language Reference*. For information on creating, editing, and debugging IC Scripts, as well as information on how to save your IC Scripts to the database, refer to *IC Database Designer Application Reference.*

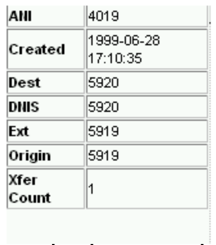
This chapter discusses the basics of the EDU Viewer, and how to create your style sheets and save them to the database. For information about adding the EDU Viewer to your Avaya Agent framework, see Chapter 4: Customizing Avaya Agent.

In this diagram, XSL stylesheets have been set using the AddTemplate API. Core receives knowledge of a new contact through an EDU. The IC Script then tells the EDU Viewer to monitor that EDU. The EDU Viewer then creates an HTML page to display in the IE browser based on the XSL stylesheet.



The EDU Viewer GUI

Here is a sample of what the EDU Viewer control might look like:



ANI	4019
Created	1999-06-28 17:10:35
Dest	5920
DNIS	5920
Ext	5919
Origin	5919
Xfer Count	1

If another contact comes into the agent's EDU viewer, then it creates a new tab showing that contact's information. Avaya Agent uses the ContactLabel property in the EDUFields section of the CDL to retrieve EDU couple that contains the tab label. Using this value in the EDU, the IC Script calls SetTabLabel for that EDU.

The XSL stylesheet

The style sheet controls how the information is displayed in the EDU Viewer.

```
<?xml version="1.0"?>
    <xsl:template match="/">
      <HTML>
      <HEAD>
      <TITLE>EDU Viewer</TITLE
      </HEAD>
      <BODY STYLE="border: 0px; margin: 0px; font-family: arial;
      font-style: bold;font-size: 10pt"
    <xsl:for-each select="vdu">
      <TABLE BORDER="1" CELLSPACING="1" CELLPADDING="1"
      STYLE="border:0px;
      margin: 0px; font-family: arial; font-style:
      bold;font-size: 8pt">
    <xsl:apply-templates/>
      </TABLE>
    </xsl:for-each>
      </BODY>
      </HTML>
    </xsl:template>
    <xsl:template match="Field">
      <TR>
    <xsl:apply-templates/>
      </TR>
    </xsl:template>
    <xsl:template match="vdu/Field">
    <xsl:if test="Name [. ='account' $or$ . ='account_value'
      $or$ . = 'ani' $or$ . = 'caller' $or$ . = 'createtime'
      $or$ . = 'cust_name' $or$ . = 'dest' $or$ . = 'question'
      $or$ . = 'dnis' $or$ . = 'ext' $or$ . = 'orig'
      $or$ . = 'purpose' $or$ . = 'transfercount'
      $or$ . = 'sender'$or$ . = 'recipient'$or$ . = 'subject'
      $or$ . = 'routingcount'or$ . = 'hasattachments']">
      <TR>
    <xsl:apply-templates/>
      </TR>
    </xsl:if>
    </xsl:template>
    <xsl:template match="vdu/Field/Container/Field">
    <xsl:if test="Name [. = 'sender' $or$ . = 'recipient'
      $or$ . = 'subject' $or$ . = 'routingcount']">
      <TR>
    <xsl:apply-templates/>
      </TR>
    </xsl:if>
    </xsl:template>
    <xsl:template match="Container">
    <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="Name">
```

```
        <TH ALIGN="left">
      <xsl:choose>
      <xsl:when test="../Name
        [. = 'account']">Account</xsl:when>
      <xsl:when test="../Name [. = 'account_value']"
        >Cust Val</xsl:when>
      <xsl:when test="../Name[. = 'ani']">ANI</xsl:when>
      <xsl:when test="../Name[. = 'caller']">Caller</xsl:when>
      <xsl:when test="../Name[. = 'createtime']">Created</xsl:when>
      <xsl:when test="../Name[. = 'cust_name']">Customer</xsl:when>
      <xsl:when test="../Name[. = 'dest']">Dest</xsl:when>
      <xsl:when test="../Name[. = 'dnis']">DNIS</xsl:when>
      <xsl:when test="../Name[. = 'ext']">Ext</xsl:when>
      <xsl:when test="../Name[. = 'orig']">Origin</xsl:when>
      <xsl:when test="../Name[. = 'purpose']">Purpose</xsl:when>
      <xsl:when test="../Name[. = 'transfercount']">
        Xfercount</xsl:when>
      <xsl:when test="../Name[. = 'sender']">From</xsl:when>
      <xsl:when test="../Name[. = 'recipient']">To</xsl:when>
      <xsl:when test="../Name[. = 'subject']">Subject</xsl:when>
      <xsl:when test="../Name[. = 'question']">Question</xsl:when>
      <xsl:when test="../Name[. = 'routingcount']">ForwardCount
      </xsl:when>
      <xsl:when test="../Name[. = 'hasattachments']"> HasAttachments
      </xsl:when>
      <xsl:otherwise>
      <xsl:value-of/>
      </xsl:otherwise>
      </xsl:choose>
        </TH>
      </xsl:template>
      <xsl:template match="Seq">
        <TH>
      <xsl:value-of/>
        </TH>
      </xsl:template>
      <xsl:template match="Value">
        <TD>
      <xsl:value-of/>
        </TD>
      </xsl:template>
</xsl:stylesheet>
```

XML script generation

For all contacts, Avaya Agent creates an EDU that has a unique ID and consists of a series of name/value fields. It is converted into the following XML structure:

```
<vdu vdu_id="3769579c000000000a64038e22bd0002">
    <Field>
       <Name>name_1</Name>
       <Value>value_1</Value>
    </Field>
    <Field>
       <Name>name_2</Name>
       <Value>value_2</Value>
    </Field>
       .
       .
       .
    <Field>
       <Name>name_N</Name>
       <Value>value_N</Value>
    </Field>
</vdu>
```

EDU containers

In addition to ordinary couples (name/value pairs), EDUs may contain hierarchical data structures organized into **containers** through field naming conventions. Containers are special EDU couples that reflect a grouping of values under a common name, and they make a tree-like data structure within the EDU. For example, these couples are containers:

`agent.1 dev6`

`agent.2 dev5`

This information reflects the agents that handled the EDU contact.

Container fields are merged into a single XML structure:

```
<Field>
     <Name>agent</Name>
     <IndexedContainer count="2" min="1" max="2">
        <Field>
           <Seq>1</Seq><Value>dev6</Value>
        </Field>
        <Field>
           <Seq>2</Seq><Value>dev5</Value>
        </Field>
     </IndexedContainer>
</Field>
```

For more information on EDUs and their contents, refer to *Electronic Data Unit Server Programmer Guide* or the *Core Services Programmer Guide*.

EDU lifecycle

There are three stages in the lifecycle of an EDU:

- Creation.
- Activity.
- Termination.

EDU creation

Every contact must have a corresponding EDU. For example, in the case of a voice contact, whenever an inbound call arrives at a telephone on the PBX system or an agent dials out, Telephony creates an EDU.

EDU activity

The EDU is a real-time storage device that collects strings of text from multiple sources and stores them as couples. Couples are paired data. During its life, the EDU's job is to collect the information entered by the contact, agent or automated software, and to notify interested clients of changes to the EDU. Typically, clients want to examine, modify, or add EDU data.

EDU termination

When an agent’s conversation ends, this does not necessarily mean that the EDU associated with the call is terminated. The agent may still need to perform wrap-up activities that involve the EDU. If the call is transferred to another agent, the call and its associated EDU continue to assist. In fact, it is possible to have multiple agents accessing a single EDU.

After an agent is finished with the EDU, the agent's interest in the EDU should be terminated. When an EDU has no more interested clients, it is usually archived to back-end databases before finally being purged from Avaya IC.

CDL settings

This component does not have any CDL settings.

Relevant integration hooks

This component has no relevant Integration Hooks.

Contact History Browser

The Contact History Browser control shows the previous contacts made by the currently selected customer. It consists of the Contact History Browser control, which is an instance of the MS Tabstrip control, and the Contact History Filter control. All three of these controls are tied together by several IC Scripts to give the user the ability to pick which customer’s contact history they want to see and how to filter that history.

The Contact History Browser displays the previous contacts made by the currently selected customer. Contact history for the customer is retrieved from IC Repository.

The Contact History Browser displays a table with the following columns:

- Time - the date and time that a particular contact was made.
- Type - the type of media that was used to make the contact (phone, email, or other).
- Subject - the subject of the previous email.

The Contact History Browser retrieves contact information from IC Repository using the DCO Application Programming Interface (API).

The Contact History Browser also caches information about contacts as they are added. In order to figure out which contact's history is being viewed, tabs are present using the MS TabStrip Control.

MS TabStrip control

The MS TabStrip control is a standard Microsoft control installed with Avaya Agent. An instance of this control is used to determine which contact is being viewed in the Contact History Browser.

Contact History Filter control

At times, the contact history that is retrieved for a particular contact is overwhelming. This information can be filtered to make it more useful to the agent. The Contact History Filter control is the filter or sorting mechanism of the Contact History Browser.

The Contact History Filter control enables agents to narrow down their search of any contacts based on the following search criteria:

- Media Type - filters the emails by the media type that was used to make the previous contact. The media type can be "phone" or "email".
- Date - displays customers contacts that occurred between two dates that are specified by the agent.
- LastNContacts - filters contacts based on the last number or contacts made by the customer whose contact history you are viewing.

Example

The three filters can be used in any combination. For example, to quickly find an email sent on July 4th:

1. Select the Media Type check box and select the email option.
2. Select the Time check box and select the Start Date and the End Date to be the July 4.

   The Contact History Browser displays a list of the emails received on the specified date.

Configuring supporting servers and databases

The Contact History Browser retrieves information about previous contacts made by the customer currently selected from IC Repository. For IC Repository to store data about previous contacts, the IC Repository database and supporting servers must be configured.

If Avaya Interaction Center was properly installed, then the supporting servers and databases will already be available. Your Avaya IC system should include:

- The IC Repository database.
- The CCQ database.
- The Report Wizard application.
- Seed data for the databases. This data includes default mapping rules for the Report server that describes how EDU data (data about a contact) should be written to the database after an EDU has been “retired.”
- The Report server.

If any of these elements are missing, refer to IC Installation and Configuration for details about how they can be installed and configured.

CDL settings

The following CDL settings are applicable to Contact History.

QSection Name	**Attribute(s)**	**Description**
CHFilter	MediaField	name of the field which is used when a media filter is applied from the Contact History filter to Contact History.
CHFilter	TimeField	name of the field which is used when a time filter is applied from the Contact History filter to Contact History.

Relevant integration hooks

The following Integration Hooks are directly related to the Contact History Browser. Use them to either enhance or change the behavior by creating Integration Hook Handlers as described in Customizing Avaya Agent behavior on page 36.

- CHBrowser_Initialize
- CHBrowser_OnContactSelected
- QConsole_BuildCHBrowserInfo

Chapter 9: Media channels

This chapter describes the three media channels in which Avaya Agent can accept contacts: voice, email, and chat.

This section includes the following topics:

The voice channel

The Avaya Softphone controls provide access to the Telephony system. This section defines some telephony terms used and describes some telephony call management concepts. This information is general in nature; see your telephone switch documentation to determine which points apply to your system.

What is softphone?

The Avaya Softphone is comprised of a set of software components that allow agents to perform telephone operations without touching a physical phone set. The Softphone enables these agents to perform standard telephony functions such as making calls, answering calls, handling conference calls, and transferring calls.

The Softphone consists of the following components:

- The VTel Automation server.
- The PhoneEngine control.
- The GUI controls.

Voice channel terms and concepts

This section defines some of the terms that are used in this section and describes a few concepts involved in telephony call management.

Phones, calls, switches

The terms **call, call type, phone line appearance**, and **switch**, as used in this chapter, are defined as follows:

- A call is an active connection between two or more parties that allows transmission of speech.
- The call type distinguishes between direct calls, which are calls placed to a specific equipment phone number, and Automatic Call Distribution (ACD) calls, which are calls placed to a phone number that is controlled by an ACD system.
- A phone relates to a phone number that can accept and/or make calls, and has at least one line appearance available. This chapter discusses simple phones and ACD phones.
    - Simple phones can accept direct calls.
    - ACD phones can accept both direct and ACD calls.
- A line appearance is the number of simultaneous calls that are possible on a phone. Available line appearances for a phone can be divided according to whether a call is inbound or outbound, internal or external, with mixed limits for different types of calls.
- A switch is a general term used to refer to a telephone switching system, such as a Private Branch Exchange (PBX) or an ACD system. The term ACD is often used in this chapter to refer to any switch or system, such as the Expert Agent Selection (EAS) on the Avaya Definity switch, capable of routing calls through a queue.

Call routing and queues

Calls can be routed in several ways:

- Calls can be routed directly to a phone when the specific phone number assigned to the physical phone set is called (a direct call).
- Calls can use a logical phone number which is mapped to the physical phone by an external resource (also considered a direct call).
- Calls can be indirectly routed through a call queue (a queued call).

A queue is a means of routing calls to any one of a number of agents qualified to handle the call. Each queue is often oriented toward a specific product, service, or skill set. In some systems, each time agents log into the system, the skills associated with their ID are used to place them in the appropriate queue. In other systems, the agent must directly specify the queue. Depending on the phone switch used, an agent can be in several queues at the same time, or can be changed to other queues by a supervisor to meet changing needs during the day.

All ACD calls are handled through a queue. When an ACD call is transferred directly to another agent (not another queue), it generally becomes a direct call.

Phone states

Some telephone switches support a variety of states for ACD phones, such as Ready, Busy, and WrapUp. They can provide statistics on the time spent in each state. Other switches may not support these states, or the rules may vary.

The Softphone configuration file is the Vtel.ini file. This configuration file accommodates and supports the variation among switches. The following is provided as a general discussion.

Note:
The terms used to describe phone states vary among call centers. For example, Ready may be known as Available, Busy as Auxiliary Work or Idle, and WrapUp as After Call Work.

An ACD phone can be in one of three states:

- **Ready** state makes the phone available to receive another call.
- **Busy** state prevents the queue from sending a new ACD call, but will generally allow new direct calls if additional inbound line appearances are available.
- **WrapUp** is a state following a call when the call has ended (on hook, hangup) but the agent is still processing information related to that call or customer.
  - When the call is in WrapUp state, it is usually implicitly busy and will not accept another ACD call, but will accept direct calls if additional inbound line appearances are available.
  - For direct calls, it is possible to set some phone switches into WrapUp mode, but this is not generally available.

While a direct call is active, you generally cannot make the phone Ready for another ACD call. Although Softphone may remain in the Ready state, the switch will prevent the queue from dispatching another call until the phone is idle.

Telephony programming overview

This section describes the Softphone components and some concepts associated with telephony programming. This section provides system level background information designed to help you better understand the Softphone.

VTel Automation server

The VTel Automation server provides the communications link between the PhoneEngine control and the Telephony servers. Telephony requests and responses are transmitted from the VTel Automation server to Softphone's ActiveX controls.

See *IC Installation Planning and Prerequisites* for supported telephone switches.

Note:
Avaya Agent does not support another application using another Vtel OLE session for telephony function calls.

PhoneEngine control

The PhoneEngine Control is an ActiveX control that gives you access to the telephony functionality that is provided by the VTel Automation server. The PhoneEngine is a non-GUI control that resides between the GUI controls and the VTel Automation server. For more information, see The GUI controls on page 84.

Note:
To successfully communicate with Telephony servers, the Phone Engine control requires that a current VTel Automation server is running in the background. If your system is running an older version of VTel, the Avaya Agent installation process automatically upgrades it to this version.

The PhoneEngine Control supplies both Telephony and CoreServices information for the logged in agent. The PhoneEngine is actually a wrapper around the following ActiveX controls:

- CoreServices control.
- Telephony Services control.

The CoreServices control provides methods and is an event source. It contains information that pertains to Login/Logout, EDU data, and Directory server information. It gets this information from the Directory server and EDU server on CTI.

The Telephony Services control provides information about the call, agent, and phone states from the Telephony server.

The CoreServices and Telephony functionality and events are the most important controls to a client developer. The PhoneEngine control contains two methods that provide access to the CoreServices and Telephony controls:

- GetCoreObject().
- GetTelephonyObject().

Each of these methods returns an IDispatch pointer, which enables you to register your client as a listener of the events that are provided. Because these are ActiveX controls, their reference count is incremented as part of each of these calls. You must perform a release when you are finished with the control.

The GUI controls

The Softphone contains the following GUI controls:

- Telephony Button.
- Call List.
- DTMF.

Telephony Button control

The Softphone contains Telephony Buttons (phone buttons) to provide telephony functionality. These buttons are enabled and disabled based on the current phone state and call state of the physical phone set and the agent state of the logged in user.

Note:
For direct calls, the WrapUp button is disabled.

The Telephony Button control component contains the following phone buttons:

Button Name	**Function**
Ready	If the phone was previously placed in the Busy or HangUp state, enables the Softphone to accept a call.
Busy	Places the Softphone in a Busy state so that it is unable to receive a queue call. When it is pressed during an active call, it places the phone in a Busy state after the call (Preset).
Answer/HangUp	Toggles between Answer and Hang Up based on the phone state. ● The Answer button is active when the phone state is Ready and the phone is ringing. ● The HangUp button disconnects the call.
Hold/Reconnect	Places a call on hold. Click again to active the Reconnect button and take the call off hold.
WrapUp	Places the Softphone in WrapUp state after the call is disconnected enabling you to enter specific information about the call.
Directory	Displays the Dial Directory window from which you can make, transfer, or conference a call.
Initiate	Enables you to make an outbound call.
Transfer	Enables you to transfer the call to another agent or queue.
ConsultativeTransfer	Allows agents to consult with each other prior to transferring the call. The agent has the ability to cancel the transfer.
Conference	Allows multiple people (three or more) to interact in a conference call.

Call List control

The Call List control displays a list of the calls that are currently assigned to the agent. For each line appearance, the Softphone displays the caller’s number, the call state, the time in call, and preset (if available).

Information Field control

The Information Field control provides the ability to display the following Telephony status information:

- Agent State.
- Phone State.
- Call State.

Control initialization

The OnBeforeLogin event that is fired from Avaya Agent is designed to be used for control initialization. The system performs the following major Softphone related tasks in this IC Script:

- Control Introduction.
- EDU Viewer Initialization.

The VTel.ini file is used to specify the switch type and other configuration items.

Registration

All of the GUI controls that are discussed in PhoneEngine control on page 84 must register with the PhoneEngine control in order to receive Telephony and/or CoreServices events. The GUI controls use the AdviseEventSinktoTelephony() method to be a listener of Telephony Services. They use AdviseEventSinktoCore() to be a listener of CoreServices.

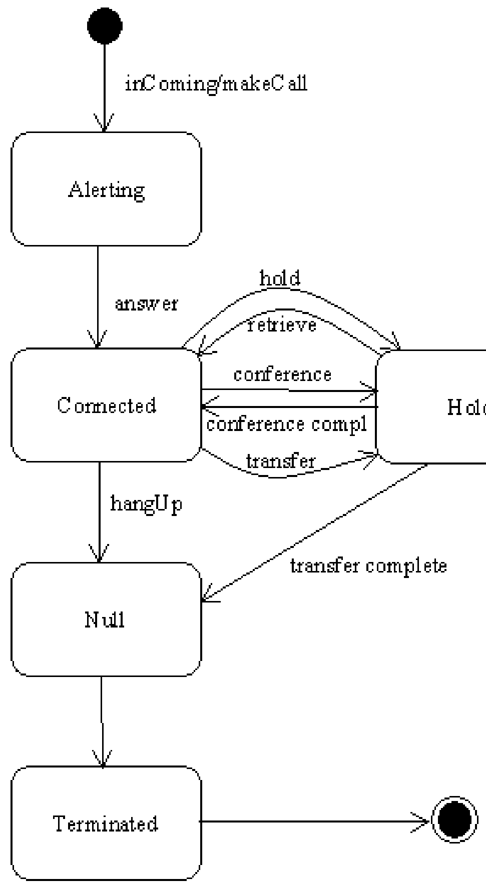
Registering as a CoreServices and Telephony listener allows each listener to receive completion events for any requests made to the controls, as well as unsolicited events such as incoming call.

The call object

Call management is done using a Call object. This object represents a single incoming or outgoing call. It provides such telephony manipulation methods as call answering, transferring, and hang-up. Call also provides methods to retrieve call state information. The EDU data is retrieved or set through the call object.

A single PhoneEngine can contain multiple Call objects — mirroring a multi-line telephone. These objects make up a PhoneEngine's call collection. All of the call specific operations are part of the Telephony control. The Telephony control provides access methods to simulate a multiple-line phone. A line can be selected and operated on before switching to another line.

Call states are illustrated in the following Call States diagram:



Accessing the EDU

Each Call object provides access to the call's EDU object. EDU data is stored as a SeqCouple (sequence of couples) object. A **couple** is a name/value attribute and can be processed as a SeqCouple, an unordered collection of zero or more Couple objects. Data is stored in the EDU as name/value pairs. Both name and value are represented as strings. Using the EDU object, the value of an attribute stored in the EDU can be retrieved directly by specifying the name of the attribute using the QCall.GetVduValue *(attribute,Value)* method.

A client can retrieve account information that a customer may have entered at the VRU by using the QCall.GetVduValue () method. This information is valuable if telephony capabilities are being integrated with other business applications.

Call termination

A Call object will be created as a result of an agent receiving an incoming call or initiating an outgoing call. However, the Call object can persist, maintaining interest in the EDU, even after the actual call has been transferred or hung up. This permits the PhoneEngine client to provide wrapup information.

Event handling

The CoreServices control emits Core events that contain a CoreChangeEvent object. The CoreChangeEvent has the following information:

- RequestObj - created and returned to the caller as part of the original request.
- VduObj - populated with EDU information or left empty if not associated with the call.
- EventId - indicates the type of event.

The Telephony control emits Telephony events that contain a ChangeEvent. The Telephony ChangeEvent has the following information:

- RequestObj - created and returned to the caller as part of the original request.
- Call- the object that contains call operations and information.
- EventId - indicates the type of event.

In order to receive events from the PhoneEngine, the client must register as a listener of Core and Telephony Events.

Note:
Both the Core and Telephony controls contain a method to return the textual representation of the eventID. It is GetName(LONG Id, BSTR *Name), the description is returned in the name field.

Each Core and Telephony Event contains an EventID to identify the type of event that occurred or the information that has changed. This information can be accessed from the get_EventId(long) method that is part of the object that arrives with the event.

Phone state

Phone state is a representation of the PhoneEngine’s interaction with the switch. The details of this interaction depend on the switch and phone type. In the direct phone case, the state model is degenerate and reflects the status of the switch association.

GUI controls

This section contains the system constants for the Information Fields and Telephony Buttons controls.

Voice event handling

In order to recognize that an agent is being assigned a call, the application must be able to process events emitted by the telephony component. While the telephony component provides call-level events such as answer, hangup, hold, un-hold, and others, not all events need to be processed by the application. Applications usually process events in one of two ways: callback or polling.

- In the callback approach, the application registers a method with the telephony component. This method is called whenever a new event occurs. Event data is passed to the application using this method.
- In the polling approach, the telephony component does not notify the application. Instead, the application must periodically query the telephony component for any event.

The telephony component also has methods to retrieve agent, phone, and call state information as integer values.

DTMF Control (Definity switch only)

The DTMF (Dual-Tone Multi-Frequency) control enables the agent to type numbers, such as the customer's account number, on a numeric key pad and sends this information as if they had dialed the tones on their telephone.

To enable support for DTMF on the Softphone, set the attribute `Visible` to `TRUE` for the "tbDTMF" control in your CDL file:

<QControl Name="tbDTMF" ProgID="QSoftPhone.TelButtonCtrl.56" Left="3" Top="128" Width="94" Height="32" Visible="TRUE"/>

CDL settings

To use the Softphone, set the following CDL parameters:

QSection Name	**Attribute(s)**	**Description**
Softphone	ButtonFontName	Name of the font used for the Softphone related buttons in Avaya Agent. Leave this blank to use the default system font.
Softphone	ButtonFontSize	Size of the font used for the Softphone related buttons in Avaya Agent.

Relevant integration hooks

The following Integration Hooks are directly related to the Voice Channel. Use them to either enhance or change the behavior by creating Integration Hook Handlers as described in Customizing Avaya Agent behavior on page 36.

- PhoneEngine_OnTelephonyStateChanged
- Shared_OnActivate
- Shared_OnSelect
- Softphone_PlaceCall

The web agent

In Avaya IC, the Web Agent is used for email and chat tasks. On the Avaya Agent pane, the ChatList control displays web tasks. The Task List control displays email tasks. There is a common control that works together in conjunction with the Web Agent and the Engines for Chat and Email. This control is the WACEngine control. It serves as the communication Hub between the WebAgentClient and Chat/Email Engines.

CDL settings

The following CDL settings are applicable to the Web Agent client:

QSection Name	Attribute(s)	Description
WACEngine	JavaVMParameters	used to pass parameters to the Java VM when the Web Agent client is instantiated by the WACEngine.

Relevant integration hooks

The following Integration Hooks are directly related to the Voice Channel. Use them to either enhance or change the behavior by creating Integration Hook Handlers as described in the Customizing Avaya Agent behavior on page 36.

- WACEngine_AgentStateChanged

- WACEngine_ErrorOccurred
- WACEngine_PerformConsoleAction

The chat channel

In order to run Chat in Avaya Agent, the Web Agent must be installed on the Agent's desktop, along with the WebEngine, ChatList, and the IC Scripts used to tie all the pieces together. (All components are installed with the Web Agent.) The following table describes these components:

Component	Description
WebEngine	ActiveX control that coordinates Web Agent, ChatList, and Avaya Agent. Is the Source for Logging in and out of Avaya Agent and receiving events from chats.
ChatList	ActiveX control that the Agent uses to select which Chat he or she is working on.

Web state event handling

The WebEngine emits the WebStateChanged event every time the state changes. This event includes an Event Object parameter, which can be interrogated to determine what type of event is occurring and what action should be taken. In the out-of-the-box Avaya Agent, the WebEngine_WebStateChanged event is assigned to this event to perform the logic needed to add the contact into Avaya Agent.

CDL settings

This component has no CDL settings.

Relevant integration hooks

The following Integration Hooks are directly related to the Voice Channel. Use them to either enhance or change the behavior by creating Integration Hook Handlers as described in Customizing Avaya Agent behavior on page 36.

- Shared_OnActivate

- Shared_OnSelect
- WebEngine_ActivateCallback
- WebEngine_TaskDeclined
- WebEngine_TransferResponse
- WebEngine_WebStateChanged

The email channel

Email Management is a software component that allows Customer Service Representatives (CSRs) to view and respond to email messages that come into the call center. It is controlled by the agent through the Web Agent and the Task List control. Using Email Management, you can:

- Receive email messages.
- View all the email messages that are currently active.
- View the details of each message.
- Defer (pend) the email.
- Transfer an email to another agent or task type.
- Choose responses using the Web Agent resources view.
- View a list of messages that have been previously received.
- View details of these historical messages.

Each Email Management component is an ActiveX control. Email Management is made up of the following controls:

- Task List.
- MailEngine.
- Web Agent.

Email management channel terms and concepts

This section defines some of the terms used in this section and describes a few concepts involved in email management.

Term	Definition
Transfer	Transfers the email to another agent or task type.
Dismiss	Dismiss the email. In Avaya Email terminology, this means to set the status of an email. Statuses are defined in the Avaya Email Manager.
Defer	Defer (pend) is the email equivalent of putting a call on hold.
Reply	Reply to an email.
History	A message which was received in the past by the call center.
CSR	Customer Service Representative. The end user of the Email Management client. May also be referred to as an agent.
WACD	The central component of the integration. This server receives messages from the email add-ons and performs the necessary actions. These actions include creating EDUs, setting EDU values, running workflows for queuing, running prompter workflows, and extracting data from Avaya data sources.

Required components

To successfully integrate Email Management, the following third party components must be installed on the system's server:

- SMTP/POP3 server – A mail server that supports the POP3 incoming mail and SMTP outgoing mail.

Email Management also requires the installation of the following Avaya IC server components:

- IC Manager – Used to administer agents and view statistics of the WACD.
- WACD server - receives messages from the email add-ons and performs the necessary actions. These actions include creating EDUs, setting EDU values, running Flows for out-of-the-box queuing, running Prompter Flows, and extracting data from Avaya data sources.
- IC Workflow server - tells the WACD server where to route the email request based on a customizable routing script.

- EDU server - creates EDUs in response to the WACD server and the client applications. It stores open EDUs, records EDU events, and provides services that enable clients to interact with an email request. The EDU server provides information like the Status, Age, and Body of the message to the Active eMail Browser.
- Report server - listens for VDU.end events and writes the information from the terminated EDU into the database. The Report server writes the EDU data directly into a normalized data model. It captures only the information that you have designated to be used in your reporting requirements.
- IC Repository - stores the historical contact information that is retrieved by the Contact History Browser. IC Repository archives terminated EDUs for long term storage that the reporting application can access to generate reports. For more information, see *OA Reports Reference.*

The Avaya Email Management client

The Email Management client is comprised of a set of components which allow an agent to quickly view a summary of email contacts:

- MailEngine control.
- Task List control.
- Web Agent.

The agent may also use the Contact History Browser to view historical information about an email. The agent can use this browser view the contents of past emails and when they were sent.

MailEngine control

The MailEngine is a non-GUI control that provides the communications link between other controls and the Avaya Telephony servers. It communicates with the Web Agent and the Core Services OCX via COM. The Web Agent is responsible for communicating with the WACD using XML. The CoreServices OCX gives the MailEngine access to the EDU server.

The MailEngine provides two major services:

- It handles events coming from the Web Agent and fires off ActiveX events that can then be handled by any component with an event sink.
- It implements the new Task interface model so that it may be used by the Task List control.

Task List control

The Task List is a GUI control that displays tasks. Although the Task List is designed to be media-independent, out-of-the-box the Task List only displays email tasks controlled by the MailEngine control. The Task List shows the state of the task, the origin, and the time in state. However, additional columns may be displayed also.

The Task List provides the following functionality:

- Allows the agent to view all email tasks currently assigned and their state. The state shown for email tasks is incoming (new), active, inactive (deferred), or wrapup.
- Allows the agent to activate an email task in the Web Agent. When activated, it becomes visible in the client.
- Allows the agent to view task-specific information in columns, a tool tip, or both.

The agent may choose to preview an email task or reply to it through the Web Agent user interface. When an agent is replying to it, it is considered to be active.

As the integrator, you can set the control object properties to display any columns of information you want, including, but not limited to, the ones shown below. The columns may be specified in the CDL by setting the value for the Columns property in the TaskList section.

The following table shows the tables that are available out-of-the-box. Those items marked “Specific to Email tasks” are examples of properties of the Mail Task object that you could also choose to display.

Column	**Out-of-the-box**	**Specific to Email tasks**
Task icon	Yes	No
State icon	Yes	No
Origin	Yes	No
Time in state	Yes	No
Subject	No	No
Age	No	No
State	No	No
EDU ID	No	No
Task ID	No	Yes
Tracking number	No	Yes
Queue time	No	Yes

For example, if you wanted to display the account balance for a task as a tooltip when the agent hovers the mouse cursor over the task, you would place the code in the initialization code of Avaya Agent, as in TaskList_Initialize.

```
Dim iQConsole As Application
Dim iTaskList As Object

Set iQConsole = GetApp
Set iTaskList = iQConsole.GetControlObject("TaskList")
iTaskList.SetFormat "{task_source_type} ({task_state}): Origin:
'{origin}' Subject: '{subject}' Tracking Number:
{[tracking_number]} Balance: {[acct_balance]}"
```

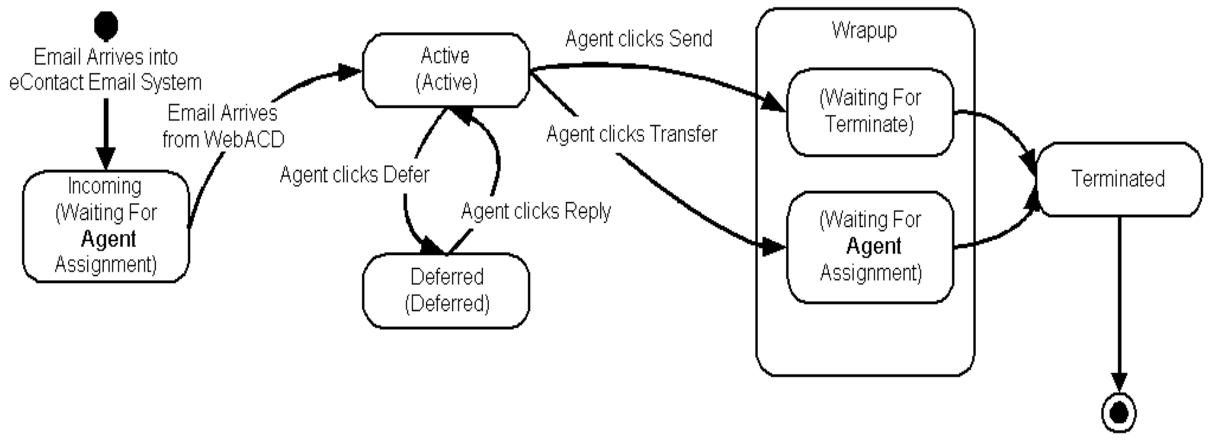
This sample code could be used to then set the account balance into the email Task object. When set, the property is displayed when the mouse is hovered over that task.

The account balance could be retrieved from an EDU object, for example.

```
Dim iQConsole As Application
Dim iTaskList As Object
Dim iProperties As Object

Set iQConsole = GetApp
Set iTaskList = iQConsole.GetControlObject("TaskList")
Set iProperties = iTaskList.Properties
iProperties.SetOneValue "acct_balance", "$245.36"
```

In this state diagram, the Mail Task states are shown. The corresponding WACD states are shown in parentheses, if applicable.



CDL settings

To configure the Task List component, you need to modify the following CDL parameters:

QSection Name	Attribute(s)	Description
Email	DeferOnLogoutWait Time	When an agent logs out, specifies the period of time (in milliseconds) that Avaya Agent waits to logout after automatically deferring the agent's email.
TaskList	Columns	Specifies the columns displayed using the Columns property.
TaskList	DisableContextMenu	Disables the context menu using the DisableContextMenu property.

For details about the QSection tag, see Customizing Avaya Agent on page 19.

Relevant integration hooks

The following Integration Hooks are directly related to the Email Channel. Use them to either enhance or change the behavior by creating Integration Hook Handlers as described in Customizing Avaya Agent behavior on page 36.

- MailEngine_TaskDeclined
- MailEngine_TransferResponse
- TaskList_Change
- TaskList_GuiActivate

Chapter 10: Prompter client integration

The Prompter client lets agents run workflow-driven scripts that assist them with handling contacts. These scripts are displayed in the Prompter client, which resides in an Avaya Agent frame (so agents can access both the script and the information in Avaya Agent's other controls). For information on writing Prompter scripts, refer to *Agent Script Workflow Reference*.

This chapter describes starting a workflow in a Prompter client instance, how workflows should be constructed to function properly, any IC Scripts that allow Prompter to work, and the CDL settings for configuring the Prompter client.

This section includes the following topics:

Starting a flow on a prompter client

You can launch a flow in a Prompter client instance using the out-of-the-box IC Script called QPrompter_StartFlow. The IC Script takes the following parameters:

- iQPrompterClient – the instance of the Prompter client in which the flow should be launched.
- sFlowserName – the name of the Flowset containing flow to be launched.
- sFlowName – the name of Flow to launch.
- FlowParameters – the parameters to be sent to the flow in an array of name/value pairs (name=value).
- sLabel – The label that appears in the Prompter client Session Tree so that the agent knows what this particular flow is designed to do.

The following portion of the IC Script's code shows how a flow can be launched using the QPrompter_StartFlow IC Script:

```
Dim iApp As Application
Dim iQPrompterClient As Object
Dim sFlowsetName As String
Dim sFlowName As String
Dim FlowParameters(1 To 3) As String
Dim sLabel As String
Dim sSessionId As String

       Set iApp = GetApp
       Set iQPrompterClient = iApp.GetActiveXControl("QPrompterClient")

       sFlowsetName = "SomeFlowsetName"
       sFlowName = "SomeFlowName"
       sLabel = "Nice Looking Label for Flow"

       FlowParameters(1) = "name1=value1"
       FlowParameters(2) = "name2=value2"
       FlowParameters(3) = "name3=value3"

       sSessionId = QPrompter_StartFlow(iQPrompterClient, sFlowsetName, sFlowName,
FlowParameters, sLabel)
```

Flow construction basics

Flow construction is the most important part of the Prompter client integration. The flow to be run must:

- Present the scripting interface
- Determine which IC Script will be used to pass the data collected in the flow to the database.

To build a flow:

1. Create a flow that an agent can run to accomplish a certain task (for more information refer to *Agent Script Workflow Reference*).
2. Set the following symbol values, within the flow, which will be extracted by the QPrompterClient_FlowDelete IC Script:
    - QcriptName – name of the IC Script to execute when the flow is done.
    - QScriptParam1, QScript Param2, ...QScriptParamN – these will become the parameters passed to the specified IC Script.
    - QScriptParamCount – Number of QScriptParam's to pass to IC Script Name

The following is a short example of how the symbols set in the flow translate into an IC Script being run at flow completion. If specifying the following symbols within the flow:

```
QScriptName="QScriptToRun"

    QScriptParam1="value1"

    QScriptParam2="value2"

    QScriptParam3="value3"
    QScriptParamCount=3
```

Then you must have created an IC Script defined as:

Sub QScriptToRun(sParam1 As String, sParam2 As String,

sParam3 As String)

CDL settings

There are no CDL settings for this component.

Relevant integration hooks

This component has no relevant Integration Hooks.

Chapter 11: Contact wrapup

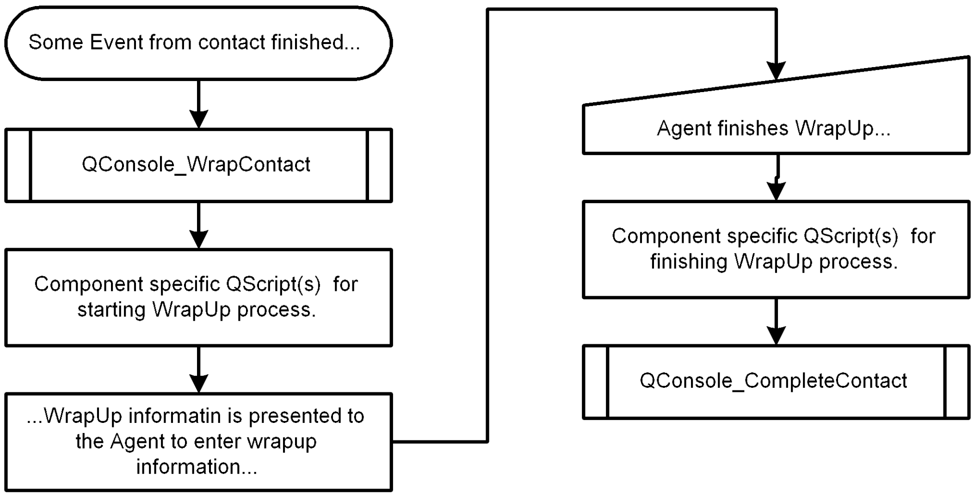
This chapter describes the out of the box wrap-up functionality included in Avaya Agent. There are two methods of performing wrap-up integrated out of the box: wrap-up through the WrapUp Dialog or WrapUp through Prompter. Both methods of wrapup are discussed in this chapter as well as the general outline of how wrapup should be handled in Avaya Agent.

When configuring wrapup, you can choose a third type of "Other". Whichever method of wrapup you use, the WrapUpEngine should be used to track wraptime as well as write the codes to Avaya IC in a way that is required by Avaya IC.

This section includes the following topics:

WrapUp process

No matter how you choose to do WrapUp in Avaya Agent, the main process is the same. The following diagram outlines this process:


Some Event from contact finished...
QConsole_WrapContact
Component specific QScript(s) for starting WrapUp process.
...WrapUp informatin is presented to the Agent to enter wrapup information...
Agent finishes WrapUp...
Component specific QScript(s) for finishing WrapUp process.
QConsole_CompleteContact

WrapUpEngine

One of the reasons for performing WrapUp when a contact ends is to gather information about that contact and the results of the agent's interaction with the contact. This information comes in the form of Wrapup Codes that are configured and built within IC Manager. More information on Wrapup Codes is provided in *IC Administration Guide*.

These wrap-up codes are then taken and written into the agent's ADU and the contact's EDU. This allows the real-time and historical reporting that keeps track of this information. Writing this data into the ADU and EDU is complex. The IC components that rely on this data, look for it in specific containers and subcontainer. Rather than force an integrator to learn this structure and implement on their own, the WrapUpEngine ActiveX control in Avaya Agent that does this for the integrator.

The WrapUpEngine doesn't just write wrap-up codes in the way IC requires it, it also provides the capability of tracking wraptime for multiple WrapUp Objects at once. This allows an integrator to use the WrapUpEngine to create custom wrap-up solutions without having to keep track of time.

WrapUpEngine API

This section describes the WrapUpEngine API.

StartWrapUp (method)

Description

This method starts a new wrap-up session in the WrapUpEngine. It creates an internal WrapUpObject.

Syntax

```
StartWrapUp(sEDUId As String, sMediaType As String)
```

Value	Description
sEDUid	The EDUID of the contact for on whom you are starting wrapup.
sMediaType	The media type string (chat, email, or voice).

FinishWrapUp (function)

Description

This function “finishes” the wrap-up session for a given EDUID. The codes are written to Avaya IC, and, if successful, the WrapUpObject is removed.

Syntax

```
FinishWrapUp(sEDUId As String) As WrapUpEngineFinishWrapUpConstants
```

Value	Description
sEDUid	The EDUID of the contact for whom you are finishing wrapup.

Returns

An integer (WrapUpEngineFinishWrapUpConstants), defined as:

```
Public Enum WrapUpEngineFinishWrapUpConstants
```

```
    wuwfcSuccess = 0
    wuwfcFailedADU = 1
    wuwfcFailedEDU = 2
End Enum
```

RemoveWrapUp (method)

Description

This method removes an internal WrapUpObject. It can be used for removing WrapUpObjects when FinishWrapUp failed.

Syntax

```
RemoveWrapUp(sEDUId As String)
```

Value	Description
sEDUid	The EDUID of the contact for whom you wish to remove the WrapupObject.

GetWrapUpObject (function)

Description

This function returns a WrapUpObject for a given EDUID. You must have previously used the StartWrapUp method to internally create this WrapUpObject.

Syntax

```
GetWrapUpObject(sEDUId As String) As WrapUpObject
```

Value	Description
sEDUid	The EDUID of the contact for whom you want to retrieve the WrapUpObject.

Returns

This function returns a WrapUpObject.

WrapUpObject API

This section describes the WrapUpObject API.

AddCodes (method)

Description

This methods lets you add a set of codes to the WrapUpObject. A set can contain 1 to 10 codes. Typically, a set contains 3 codes (category, outcome, and reason).

Syntax

```
AddCodes(sCodeKey1 As String, Optional sCodeKey2 As String, _
         Optional sCodeKey3 As String, Optional sCodeKey4 As String, _
         Optional sCodeKey5 As String, Optional sCodeKey6 As String, _
         Optional sCodeKey7 As String, Optional sCodeKey8 As String, _
         Optional sCodeKey9 As String, Optional sCodeKey10 As String)
sCodeKey<N> - This is a string code
```

EDUId (read-only property)

Description

This is a read-only property that returns the EDUId of the WrapUpObject.

Syntax

```
EDUId() As String
```

MediaType (property)

Description

This property allows you to get/set the media type for the WrapUpObject.

Syntax

```
MediaType() As String
```

WrapTime (property)

Description

This property allows you to get/set the wraptime (in seconds) for a contact. During a wrap-up session, the WrapTime accumulates. However, if you set the WrapTime explicitly, it is stored. If you set WrapTime to an empty string, WrapTime will dynamically accumulate again.

Syntax

```
WrapTime() As String
```

WrapUpDialog Wrap-up

The Wrap Up Dialog may be used when an agent needs to complete a contact by indicating a wrapup **reason**, **category**, and **outcome**:

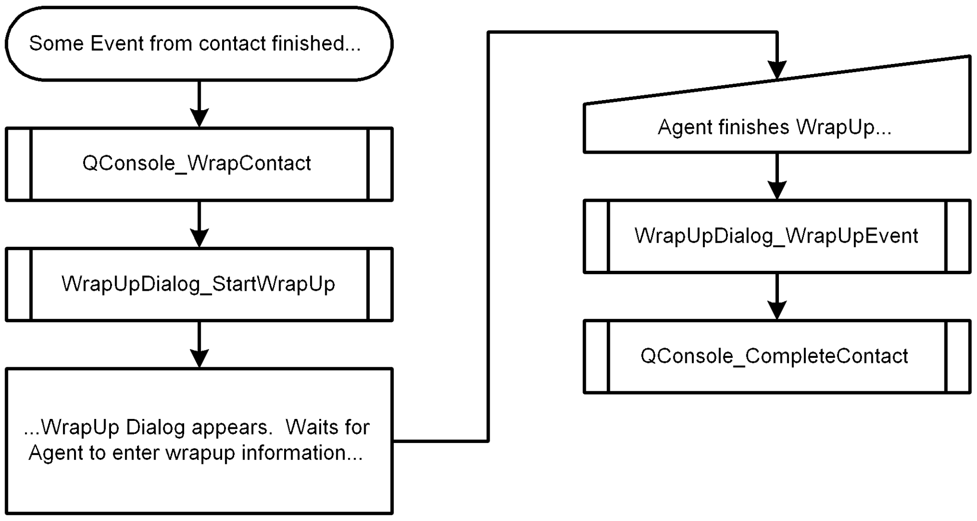
- Reason – lets the agent to specify the purpose of a customer's initial reason for contacting that agent, or the intent of the work performed (for example, Account Balance, Fund Transfers).
- Category – lets the agent wrap up more efficiently by presenting only the categories associated with that reason (for example, Insurance: Auto, Home, Life).
- Outcome – specifies the result of the contact with the customer.

Multiple WrapUp Dialogs may be shown at a time. For example, there may be wrapup needed on both the chat and e-mail channels at the same time. This lets the designer invoke the Wrapup Dialog method ShowWrapup followed by WrapupContact while another wrapup operation is still in progress.

If a contact is transferred between agents, each agent may select a wrapup reason. Each of these wrapup reasons may be shown in a list that shows all the previous entries made by other agents.

WrapUp process with the wrap-up dialog

As stated at the beginning of this chapter, whatever method is used to perform the WrapUp in Avaya Agent, the process remains the same. The following diagram shows how the WrapUp Dialog fits into this process:

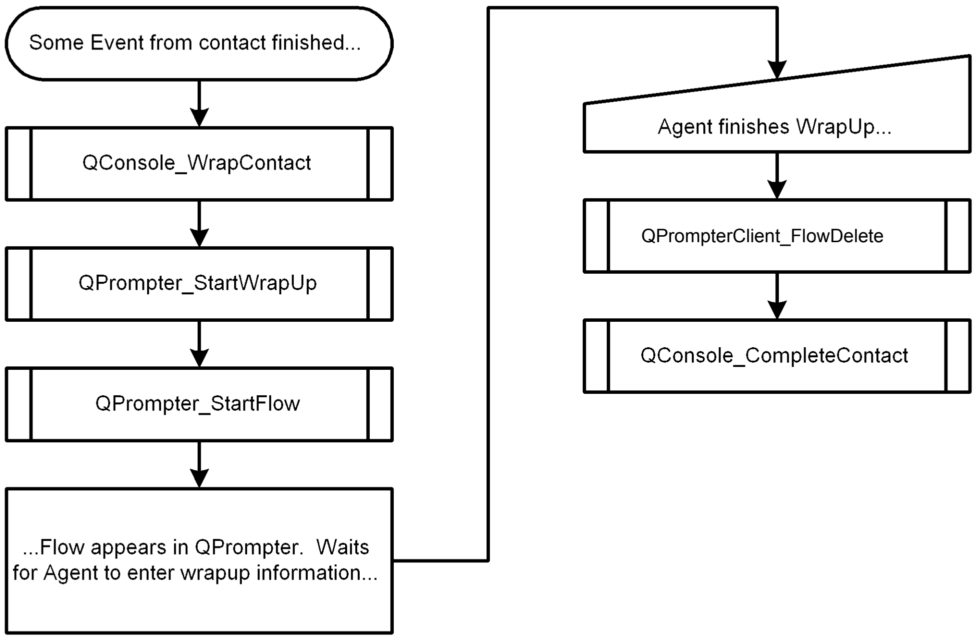


Prompter wrap-up

This section outlines how the Prompter client performs Contact WrapUp. As described in Flow construction basics on page 100, the most important part of using the Prompter client is designing the proper flow. When creating the flow for wrap-up, you will need to devise a script for the Agent to follow that results in a set of wrap-up data. You can write your data to a database, EDU, or a 3rd Party system. This can be done on the Avaya Work Flow Designer or the Avaya Agent. Because doing wrap-up through Prompter is so generic, the Integrator does most of the tasks. The framework that is provided out-of-the-box gives you a lot of flexibility for your implementation.

WrapUp process using the Prompter client

As stated at the beginning of this chapter, whatever method is used to perform the WrapUp in Avaya Agent, the process remains the same. The following diagram shows how the Prompter client fits into this process:


Some Event from contact finished...
QConsole_WrapContact
QPrompter_StartWrapUp
QPrompter_StartFlow
...Flow appears in QPrompter. Waits for Agent to enter wrapup information...
Agent finishes WrapUp...
QPrompterClient_FlowDelete
QConsole_CompleteContact

Setting the correct IC Script variables

As described in Starting a flow on a prompter client on page 99, there are several global variables that need to be set in a Prompter flow if an IC Script needs to be run. Because we need to remove the contact from Avaya Agent when WrapUp is complete, these parameters need to be set up to run QConsole_CompleteContact. The following are required settings in the Prompter flow to complete the contact in Avaya Agent properly:

- QScriptName = "QConsole_CompleteContact"
- QScriptParam1 = eduid (from input parameters)
- QscriptParam2 = mediatype (from input parameters)
- QScriptParamCount = “2”

Other pointers

As described in Contact wrapup on page 103, doing Prompter Wrap-up is primarily a customization exercise. This section provides the following pointers for doing various things.

If you want to write wrap-up codes and wraptime to the ADU/EDU, you should use the WrapUpEngine. To do this, you can use the QConsole_WrapContact Integration Hook to start a wrap-up session on the WrapUpEngine (which, is included in the out-of-the-box .cdl layout). You can then change the Prompter flow to set your own IC Script name to run when the flow is done, passing data from the flow into your IC Script. Use this Script to take the data, determine the applicable wrapcodes, and use the WrapUpEngine API store the codes and finish the wrap-up.

Other wrap-up

As described in the Contact wrapup on page 103, there is another type of wrap-up possible in Avaya Agent, which is named “Other”. Setting Agent/Desktop.WrapUpType to “Other”, gives you the responsibility of handling the wrap-up. When the WrapUpType is “Other”, Avaya Agent stops after the “QConsole_WrapContact” point in the Wrap-up Process and waits for some “other” source to run the “QConsole_CompleteContact”.

This puts how to do wrap-up, what you do during wrap-up, and even if you want to do wrap-up, totally in your control. You can take advantage of the QConsole_WrapContact Integration Hook to trigger your own wrap-up processing. You must call “QConsole_CompleteContact” when you are finished.

With this option, you can implement Selective After Contact Work….simple wrap-up button wrap-up…3rd Party Application initiated wrap-up.

Relevant integration hooks

The following Integration Hooks are directly related to WrapUp. Use them to either enhance or change the behavior by creating Integration Hook Handlers as described Customizing Avaya Agent behavior on page 36.

- QConsole_WrapContact

Appendix A: Troubleshooting

This appendix provides information about common errors and error messages that may occur when using and modifying Avaya Agent. Each error is accompanied by description, possible problem(s), and corresponding possible solution(s).

The errors covered in this document are:

The type of messages logged is controlled by the `DebugLevel` setting in Application Preferences, stored in the Avaya database. If `DebugLevel` is set to 1, 2, or 3, Avaya IC logs any errors that Avaya Agent encounters, along with all Login and Shutdown messages, to the application log file stored in *`IC_INSTALL_DIR`*`\IC73logs`. In addition, if the DebugLevel is set to 3, the messages from every IC Script that Avaya Agent runs are also logged. (For more information, see *IC Administration Guide*.)

Extra VTel session popping up when logging onto Avaya Agent

Set GuiType to none in the bin directory.

The modified eduviewer layout is not showing up in the EDU window in Avaya Agent.

Solution: Deleted all the layouts from the database, and deleted the adl, adf, and cdl from the apps directory. Log back in.

Cannot enter a non-US style phone number in agent desktop application

The data mask for phone number format is fixed at the time of installation, and cannot be modified within an application. You can remove or modify the masks using Database Designer by editing the object of the form in question.

Cannot log into components with an empty password

Avaya Agent was designed so that if a user has synchronized all of their passwords all components contained within will be logged into from the main Avaya Agent Login Tab. To accomplish this, the following logic was built into the OOB IC Scripts for Avaya Agent.

For each component to log into, get the Login ID & Password from the corresponding tab in the Avaya Agent Login. If any Login ID and/or Password was left blank, then fill with the default from Avaya Agent Login.

Therefore, if a User actually has a Login ID with an empty Password, they may not be able to log into that component because the default will be used from the Avaya Agent login. There are two ways this can be handled. The Developer can modify the IC Script logic to not apply defaults to Passwords OR always have a password for a Login ID.

Database login failed. Please retry.

This happens when Avaya Agent could not log into the database. Could be caused by:

- This happens when an invalid Login ID/Password combination is used for Avaya Agent. Use correct Login ID/Password.
- The Data server is down. Make sure the Data server is started.
- The Database is down. Make sure the Database is up.
- The IC Data Source was specified incorrectly

Error occurred at Line x, Column n while parsing CDL file

When pushing a Avaya Agent layout specification to the database, the specification is parsed to make sure it follows basic XML structure rules. If there is anything fundamentally wrong with the `.cdl` file, the above error will occur.

To resolve the problem, note the Line number, open your .cdl file and look for a fundamental syntax problem. This is usually caused by an unmatched starting and ending tag. For example:

```
<QControl Name="PhoneEngine" ProgID="QPHONEENGINE.QPhoneEngineCtrl.1">

     <QScriptDictionary>

        <QScript Event="OnTelephonyStateChanged"

Name="Softphone_OnTelephonyStateChanged"/>

</QControl>
```

The above example would cause a parsing error because the `<QScriptDictionary>` is missing a matching `</QScriptDictionary>` after it. The below example is how this error would be fixed:

```
<QControl Name="PhoneEngine"

ProgID="QPHONEENGINE.QPhoneEngineCtrl.1">

     <QScriptDictionary>

        <QScript Event="OnTelephonyStateChanged"
Name="Softphone_OnTelephonyStateChanged"/>

     </QScriptDictionary>
</QControl>
```

Avaya Agent Error 11017: Cannot find layout – xxxxxx …

Avaya Agent uses the layout file (.cdl) to determine how it looks. It pulls this file from the database upon startup. There are two places in which the name for which specification to use is determined. First and foremost, there is a parameter in the command line for starting Avaya Agent (-layout). Second, when Avaya Agent starts up, it calls an IC Script Hook called `AfterLoginHook` (located in `system.qsc`). Inside this IC Script, a developer can change which specification name to be used by changing the value of a variable `sLayoutName`. When `AfterLoginHook` completes, Avaya Agent uses the end layout name and searches the database for the specification to pull onto the local machine. This being said, there are two reasons why you would get the above error message:

- The most basic is the developer forgot to push the layout specification to the database. Use Database Designer and push the specification using "Generate Windows Application…"/Avaya Agent Layout.
- The name in the specification pushed to the database does not match the one Avaya Agent was looking for. The name appearing in the error message should match the name setting in the CDL file:

```
<QConsole Name="avaya_agent_en" Version="7.3" Description="Default
Avaya Agent Layout Spec in English">
```

In the above example, the name of the specification is "qconsole", but let us assume that the error message shows that Avaya Agent was looking for "qconsolespec". To fix this, either the name has to be changed, or the name Avaya Agent was given to look for must be changed.

Note:
Before Avaya Agent searches the database, it translates the specification name to lower case. That means the name is case-insensitive.

Finally, Avaya Agent specifications are pushed to the database per application. For example if Avaya Agent is hooking up to a ccq_request design, make sure you did not push the layout specification to the ccq_contact application.

Softphone Logout Failed: xx

If an agent is using Avaya Agent containing Softphone, when Avaya Agent shuts down, it will attempt to log that agent out of the Softphone. If, for some reason, Avaya Agent cannot log the agent out of the Softphone, they will be presented with the above message. xx will usually give you some indication of why Avaya Agent could not log out. The most common cause of this is the agent having a call in the Softphone when they try to logout.

The Avaya Agent configuration has changed Please re-log into all components

As mentioned above, Avaya Agent is built base on a layout specification. Not only is Avaya Agent's layout determined by this specification, but so is the Login dialog that appears. Therefore, if a developer pushes a new layout specification to the database, the Login dialog could have possibly changed. Thus Avaya Agent recognizes this and will force the user to "…re-log into all components…"

<name> Login Failed: <reason> Do you want to Retry?

Most operations done in Avaya Agent are done via IC Scripts, including logging into the components and controls. Avaya Agent uses standard login messages so that you can modify them to fit your needs. The above error message is specific to the Softphone, but it can be applied to other components and controls as well.

When you encounter one of these error messages, there are two pieces of information that should tell you what failed and why:

- <name> – Component/Control name whose login failed.
- <reason> – Reason why login failed.

Avaya Agent exits without restoring the desktop area it occupied

Resize the system task bar to reclaim the desktop area.

A visible control is in the CDL, but does not appear in Avaya Agent

- Make certain the control's Visible property is set to "TRUE".
- Make certain the control is positioned so that it can be seen. On lower resolution video modes, you may not see the whole control (for example, the Chat List control)

“guest_xyz@company.com” appears in the Origin field of the Email Task List

- Check WACD function as described in the IC Installation and Configuration.
- Check that WACD hostname is set up properly, via WACD Administration

Avaya Agent hangs on startup

- Make certain that you have rebooted after installing Avaya Agent.
- Check that "EnhancedDialDirectory" and "FilterADUOLE" are set to "y" in the `IC_INSTALL_DIR\IC73bin\vtel.ini` file.

Get "Warning: Could not get EDU for incoming contact"

For Email contacts:

1. Check that the WACD is working correctly.
   - In IC Manager, verify that the WACD is running.
   - Check the WACD log file (in `IC_INSTALL_DIR\IC73logs`) to see if there are errors that may indicate the cause of the problem.
2. Check that the WACD hostname is set up properly in the Email Script. This value is set in the WACD administration pages. For more information, refer to *IC Installation and Configuration.*
3. Check if the EDU timed out.
   - Using IC Manager's DDE Direct, invoke `"[VDU.GetValues("eduid")]"`, where eduid is the EDUID that may have timed out. This should return the EDU's contents if things are working properly.
   - Check the agent's `vagent_vtel.log` file for a line `"[VDU.GetValues("eduid")]"` and see if it returns the EDU successfully.

   If either check failed, the DUStore may be failing. Check that the DUStore is properly configured and running.

For Web Management chat:

- Try changing the `IC_INSTALL_DIR\IC73chatserver\website\public\escalate.jsp` ChatConnector timeout. For more information, refer to *IC Installation and Configuration*.

For Web Management chat transfers:

- Check that you have a DUStore server. Set up the DUStore in IC Manager as a server. If DUStore is not set up properly and the EDU server is shut down, the EDU server will not be able to restore the EDU from the DUStore.

Get “Invalid Email Server” from Web Agent when starting Avaya Agent

- Check that the Email server is running. In IC Manager check to see that the Email server is Up.
- Check that the WACD Email is configured properly. For more information refer to IC Installation and Configuration.

Avaya Agent customization issues

If a catastrophic failure occurs with any Avaya IC CORBA customized server from which it does not recover, dependent custom desktop components will attempt to communicate with the failed server. To avoid desktop delays, it is recommended that the custom component and server support Assigns to bind the desktop to a particular instance for the life of the session. Do not perform any MakeRequestSynch() requests on Core Services if not logged in to Core Services. This will peg the CPU.

The Out of string space error

Avaya IC scripts sometimes displays an error message: `Out of string space`. Following are the scenarios in which the Avaya IC script displays the error message:

The length of a string exceeds 32 KB

In this scenario, check all the string concatenations in the IC scripts.

For example:

```
If (len(debugout) + len({_#vduData}(i).name) + _
numSpaces + len({_#vduData}(i).value) + 3) < {_#maxDebugLength} then
debugOut = debugOut & {_#vduData}(i).name & ":" & _
String$(numSpaces," ") & {_#vduData}(i).value & ebCrLf
```

Where, `maxDebugLength` is equal to 32000.

Number of strings is greater than 32 KB

In this scenario, check the IC scripts for the number of string used. Try to reduce the number of strings used in the IC scripts.

The total size of all the strings exceeds 32 MB

In this scenario, check the IC scripts for the number of string used and also the length of each string. Try to reduce both, the number of strings and the length of each string used in the IC scripts.

Index