Genesys Desktop 7.6

.NET Toolkit

Developer's Guide

About Genesys

Genesys Telecommunications Laboratories, Inc., a subsidiary of Alcatel-Lucent, is 100% focused on software for call centers. Genesys recognizes that better interactions drive better business and build company reputations. Customer service solutions from Genesys deliver on this promise for Global 2000 enterprises, government organizations, and telecommunications service providers across 80 countries, directing more than 100 million customer interactions every day. Sophisticated routing and reporting across voice, e-mail, and Web channels ensure that customers are quickly connected to the best available resource—the first time. Genesys offers solutions for customer service, help desks, order desks, collections, outbound telesales and service, and workforce management. Visit www.genesyslab.com for more information.

Each product has its own documentation for online viewing at the Genesys Technical Support website or on the Documentation Library DVD, which is available from Genesys upon request. For more information, contact your sales representative.

Notice

Although reasonable effort is made to ensure that the information in this document is complete and accurate at the time of release, Genesys Telecommunications Laboratories, Inc., cannot assume responsibility for any existing errors. Changes and/or corrections to the information contained in this document may be incorporated in future versions.

Your Responsibility for Your System's Security

You are responsible for the security of your system. Product administration to prevent unauthorized use is your responsibility. Your system administrator should read all documents provided with this product to fully understand the features available that reduce your risk of incurring charges for unlicensed use of Genesys products.

Trademarks

Genesys, the Genesys logo, and T-Server are registered trademarks of Genesys Telecommunications Laboratories, Inc. All other trademarks and trade names referred to in this document are the property of other companies. The Crystal monospace font is used by permission of Software Renovation Corporation, www.SoftwareRenovation.com.

Technical Support from VARs

If you have purchased support from a value-added reseller (VAR), please contact the VAR for technical support.

Technical Support from Genesys

If you have purchased support directly from Genesys, please contact Genesys Technical Support at the following regional numbers:

Region	Telephone	E-Mail
North and Latin America	+888-369-5555 or +506-674-6767	support@genesyslab.com
Europe, Middle East, and Africa	+44-(0)-1276-45-7002	support@genesyslab.co.uk
Asia Pacific	+61-7-3368-6868	support@genesyslab.com.au
Japan	+81-3-6361-8950	support@genesyslab.co.jp

Prior to contacting technical support, please refer to the *Genesys Technical Support Guide* for complete contact information and procedures.

Ordering and Licensing Information

Complete information on ordering and licensing Genesys products can be found in the ***Genesys 7 Licensing Guide*.**

Released by

Genesys Telecommunications Laboratories, Inc. www.genesyslab.com

Document Version: 76gd_dev_dotnet-toolkit_07-2008_v7.6.101.00

Table of Contents

Preface

Welcome to the *Genesys Desktop 7.6 .NET Toolkit Developer's Guide*. This document introduces you to the concepts, terminology, and procedures relevant to the Genesys Agent Desktop .NET Toolkit 7.6.

This document is valid for all the 7.6.x release(s) of this product.

Note: For versions of this document created for other releases of this product, please visit the Genesys Technical Support website, or request the Documentation Library DVD, which you can order by e-mail from Genesys Order Management at orderman@genesyslab.com.

This preface provides an overview of this document, identifies the primary audience, introduces document conventions, and lists related reference information:

Genesys Agent Desktop .NET Toolkit 7.6 is a development toolset for building a rich custom agent desktop application based on .NET technology.

Genesys Agent Desktop .NET Toolkit components provide agent access to Multimedia services including inbound voice, outbound services, e-mail, chat, callback interactions, and open media interactions.

Intended Audience

This guide, primarily intended for developers, assumes that you have a basic understanding of:

- Computer-telephony integration (CTI) concepts, processes, terminology, and applications
- Network design and operation

- Your own network configurations
- .NET framework
- Client -server architectures
- XML

You should also be familiar with Genesys Framework architecture and functions.

Usage Guidelines

The Genesys developer materials outlined in this document are intended to be used for the following purposes:

- Creation of contact-center agent desktop applications associated with Genesys software implementations.
- Creation of a specialized client application specific to customer needs.

The Genesys software functions available for development are clearly documented. No undocumented functionality is to be utilized without Genesys's express written consent.

The following Use Conditions apply in all cases for developers employing the Genesys developer materials outlined in this document:

1. Possession of interface documentation does not imply a right to use by a third party. Genesys conditions for use, as outlined below or in the *Genesys Developer Program Guide,* must be met.
2. This interface shall not be used unless the developer is a member in good standing of the Genesys Interacts program or has a valid Master Software License and Services Agreement with Genesys.
3. A developer shall not be entitled to use any licenses granted hereunder unless the developer's organization has met or obtained all prerequisite licensing and software as set out by Genesys.
4. A developer shall not be entitled to use any licenses granted hereunder if the developer's organization is delinquent in any payments or amounts owed to Genesys.
5. A developer shall not use the Genesys developer materials outlined in this document for any general application development purposes that are not associated with the above-mentioned intended purposes for the use of the Genesys developer materials outlined in this document.
6. A developer shall disclose the developer materials outlined in this document only to those employees who have a direct need to create, debug, and/or test one or more participant-specific objects and/or software files that access, communicate, or interoperate with the Genesys API.

7. The developed works and Genesys software running in conjunction with one another (hereinafter referred to together as the "integrated solutions") should not compromise data integrity. For example, if both the Genesys software and the integrated solutions can modify the same data, then modifications by either product must not circumvent the other product's data integrity rules. In addition, the integration should not cause duplicate copies of data to exist in both participant and Genesys databases, unless it can be assured that data modifications propagate all copies within the time required by typical users.
8. The integrated solutions shall not compromise data or application security, access, or visibility restrictions that are enforced by either the Genesys software or the developed works.
9. The integrated solutions shall conform to design and implementation guidelines and restrictions described in the *Genesys Developer Program Guide* and Genesys software documentation. For example:
   a. The integration must use only published interfaces to access Genesys data.
   b. The integration shall not modify data in Genesys database tables directly using SQL.
   c. The integration shall not introduce database triggers or stored procedures that operate on Genesys database tables.

Any schema extension to Genesys database tables must be carried out using Genesys Developer software through documented methods and features.

The Genesys developer materials outlined in this document are not intended to be used for the creation of any product with functionality comparable to any Genesys products, including products similar or substantially similar to Genesys's current general-availability, beta, and announced products.

Any attempt to use the Genesys developer materials outlined in this document or any Genesys Developer software contrary to this clause shall be deemed a material breach with immediate termination of this addendum, and Genesys shall be entitled to seek to protect its interests, including but not limited to, preliminary and permanent injunctive relief, as well as money damages.

Chapter Summaries

In addition to this preface, this document contains the following chapters:

- Chapter 1, "About Genesys Desktop .NET Toolkit," on page 13, introduces the Genesys Desktop 7.6 .NET Toolkit with an overview of it design features along with the structure and key concepts of the library API.
- Chapter 2, "About the Examples," on page 27, introduces the supplied source code examples.
- Chapter 3, ".NET Toolkit Examples," on page 41, discusses the consistent explanation about the GUI Examples delivered with this developer's guide.

Document Conventions

This document uses certain stylistic and typographical conventions—introduced here—that serve as shorthands for particular kinds of information.

Document Version Number

A version number appears at the bottom of the inside front cover of this document. Version numbers change as new information is added to this document. Here is a sample version number:

gd_dev_dotnet-toolkit _09-2007_v7.6.000.01

You will need this number when you are talking with Genesys Technical Support about this product.

Type Styles

Italic

In this document, italic is used for emphasis, for documents' titles, for definitions of (or first references to) unfamiliar terms, and for mathematical variables.

Examples:

- Please consult the *Genesys 7 Migration Guide* for more information.
- *A customary and usual practice* is one that is widely accepted and used within a particular industry or profession.
- Do *not* use this value for this option.
- The formula, $x + 1 = 7$ where x stands for . . .

Monospace Font

A monospace font, which looks like `teletype or typewriter text`, is used for all programming identifiers and GUI elements.

This convention includes the *names* of directories, files, folders, configuration objects, paths, scripts, dialog boxes, options, fields, text and list boxes, operational modes, all buttons (including radio buttons), check boxes, commands, tabs, CTI events, and error messages; the values of options; logical arguments and command syntax; and code samples.

Examples:

- Select the `Show variables on screen` check box.
- Click the `Summation` button.
- In the `Properties` dialog box, enter the value for the host server in your environment.
- In the `Operand` text box, enter your formula.
- Click `OK` to exit the `Properties` dialog box.

- The following table presents the complete set of error messages T-Server® distributes in `EventError` events.
- If you select `true` for the `inbound-bsns-calls` option, all established inbound calls on a local agent are considered business calls.

Monospace is also used for any text that users must manually enter during a configuration or installation procedure, or on a command line:

Example:

- Enter `exit` on the command line.

Screen Captures Used in This Document

Screen captures from the product GUI (graphical user interface), as used in this document, may sometimes contain a minor spelling, capitalization, or grammatical error. The text accompanying and explaining the screen captures corrects such errors *except* when such a correction would prevent you from installing, configuring, or successfully using the product. For example, if the name of an option contains a usage error, the name would be presented exactly as it appears in the product GUI; the error would not be corrected in any accompanying text.

Square Brackets

Square brackets indicate that a particular parameter or value is optional within a logical argument, a command, or some programming syntax. That is, the parameter's or value's presence is not required to resolve the argument, command, or block of code. The user decides whether to include this optional information. Here is a sample:

```
smcp_server -host [/flags]
```

Angle Brackets

Angle brackets indicate a placeholder for a value that the user must specify. This might be a DN or port number specific to your enterprise. Here is a sample:

```
smcp_server -host <confighost>
```

Related Resources

Consult these additional resources as necessary:

- *Genesys Integration Server 7.6 Deployment Guide,* which details important configuration data.
- CHM API references, which are located in the `doc/` subdirectory within the product installation directory tree.

- *Genesys Voice Callback 7.6 Deployment Guide,* which provides configuration information for the Voice Callback Solution.
- The *Genesys Technical Publications Glossary,* which ships on the Genesys Documentation Library DVD and which provides a comprehensive list of the Genesys and CTI terminology and acronyms used in this document.
- The *Genesys 7 Migration Guide*, also on the Genesys Documentation Library DVD, which provides a documented migration strategy from Genesys product releases 5.1 and later to all Genesys 7.x releases. Contact Genesys Technical Support for additional information.
- The Release Notes and Product Advisories for this product, which are available on the Genesys Technical Support website at http://genesyslab.com/support.
- Information on supported hardware and third-party software is available on the Genesys Technical Support website in the following documents:
- *Genesys 7 Supported Operating Systems and Databases*
- *Genesys 7 Supported Media Interfaces*

Genesys product documentation is available on the:

- Genesys Technical Support website at http://genesyslab.com/support.
- Genesys Developer website at http://devzone.genesyslab.com.
- Genesys Documentation Library DVD, which you can order by e-mail from Genesys Order Management at orderman@genesyslab.com.

Making Comments on This Document

If you especially like or dislike anything about this document, please feel free to e-mail your comments to Techpubs.webadmin@genesyslab.com.

You can comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this document. Please limit your comments to the information in this document only and to the way in which the information is presented. Speak to Genesys Technical Support if you have suggestions about the product itself.

When you send us comments, you grant Genesys a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

Chapter

1 About Genesys Desktop .NET Toolkit

This chapter introduces the Genesys Desktop .NET Toolkit. It contains the following topics:

Overview

The Genesys Desktop .NET Toolkit provides a developer tools for creating .NET-compliant, agent-facing desktop client applications. To create client applications, you can use the Genesys Desktop .NET Toolkit's GUI components for rapid application development (RAD) using drag-and-drop.

The Genesys Desktop .NET Toolkit also provides an entry point to the Genesys services lower-level API to create specific applications, or to integrate the services into other application.

The Genesys Desktop .NET Toolkit lets you develop .NET applications for the following purposes:

- Create a contact center agent desktop application to let agents interact with Genesys software.

- Create an application that integrates third-party software with Genesys software.
- Create other applications specific to your needs.

See “Scope of Use” on page 16 for further details on the .NET Toolkit features.

The Genesys Desktop .NET Toolkit relies on Agent Interaction Services which send requests to a server-side application that provides the services’ features. The Genesys Integration Server (GIS) hosts this server-side application, as presented in Figure 1.

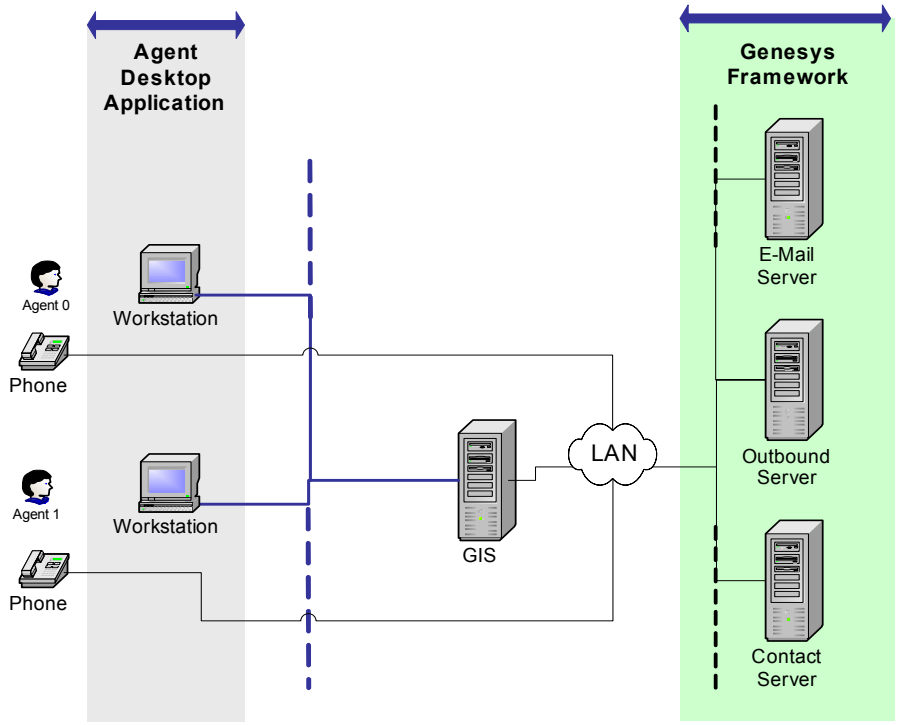

Figure 1: A .NET Toolkit Application Working in a LAN with GIS

You use the Genesys Desktop .NET Toolkit to develop client-side applications. These client-side applications offer the agent a GUI desktop. Applications built on top the .NET Toolkit can either use SOAP or GSAP protocols to communicate with GIS that interacts with the Genesys Framework.

Components

The Genesys Desktop .NET Toolkit comprises the following:

- Assemblies:
  - `AgentDesktopToolkit.dll`—this library provides the drag-and-drop GUI assembly for rapid application development.
  - `AILLibrary.dll`—this library provides the Agent Interaction Service Proxy Library for .NET, designed for exacting application requirements.

  - `AilServicesPropProtocol.dll`—this library provides the Agent Interaction GSAP Library for .NET.
  - The `ail-configuration.xml` file—configuration file used with the Agent Interaction Service Proxy Library for .NET.

> **Note:** These assemblies are available in the `<installation_directory>` directory of the Genesys Desktop .NET Toolkit Multimedia Application Sample.

- An API reference in Compiled HTML format, covering the .NET Toolkit API.
- A set of examples. For further information, see Chapter 2, "About the Examples," page 27.

As presented in the previous sections, the Genesys Desktop .NET Toolkit presents a high-level API in the form of a broad range of components available in the Visual Studio toolbox.

This set of components lets you manage agent and interaction features, as well as such services as voice telephony, outbound campaigns, and callback.

These drag-and-drop components completely abstract the underlying details of the services, so events and requests are transparently handled. However, these GUI components provide an entry point to the underlying Agent Interaction Services API so you can access specific-service details to fine-tune your application.

Architecture

The Genesys Desktop .NET Toolkit makes use of the lower Agent Interaction Service layer and includes complete interactive GUI components abstracting the implemented services.

To connect GIS that hosts the Service Core, the Agent Interaction Service layer provides either GSAP or SOAP depending on the library used to run the Genesys Desktop .NET Toolkit (See "Components" on page 14). This application might be available, for example, on Genesys Integration Server (GIS), which integrates the exposed services.

Figure 2 on page 16 presents the architecture of the Genesys Desktop .NET Toolkit communicating with GIS.

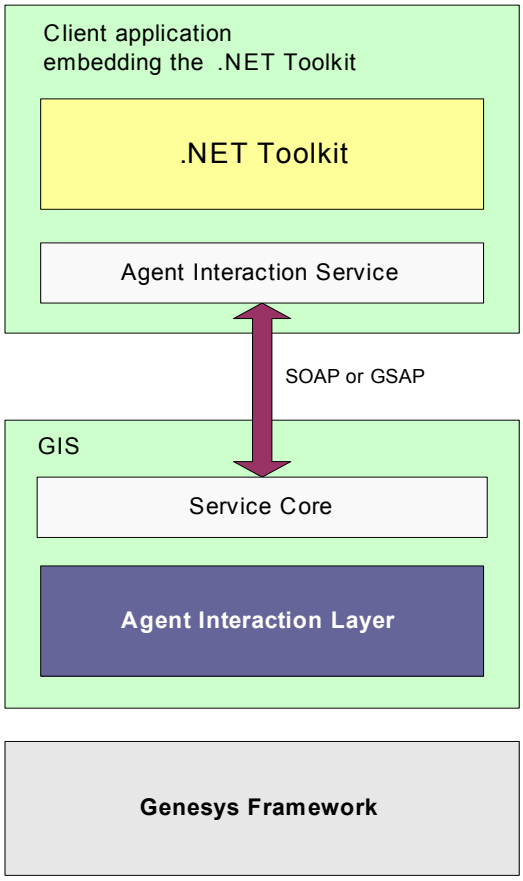

Figure 2: Architectural Overview

Figure 2 shows that the Agent Interaction Service layer communicate with the corresponding exposed services of GIS. The communication is performed through the SOAP or GSAP protocols, available for both client and server side applications.

On the server side, the services integrate the Agent Interaction Layer (AIL), which is a Genesys library available in the Genesys Interaction SDK. This library enables the Service Core to deal with the Genesys Framework and to perform the client-side services' requests.

Scope of Use

The Genesys Desktop .NET Toolkit typical usage scenarios include:

- Managing agent activity:
  - Login, logout.
  - Ready, Not-ready, and After-call-work features.

- Handling voice interactions (depending on your switch's available features):
  - Make an outgoing call,
  - Answer an incoming call,
  - Hold and retrieve a call,
  - Transfer a call,
  - Alternate calls,
  - Initiate, enter and leave a conference.
- Handling the callback feature:
  - Accept, reject, or cancel a request,
  - Accept and dial a callback request,
  - Reschedule the record.
- Handling e-mail:
  - Create and send an e-mail,
  - Reply an e-mail,
  - Transfer an e-mail,
  - Pull an e-mail off a workbin.
- Handling outbound campaigns:
  - Add a new record to the campaign,
  - Request a record,
  - Cancel a record,
  - Reject a record.
- Using the Standard Response Library:
  - Get standard responses,
  - Get standard response information about categories,
  - Get standard responses by category,
  - Manage favorite standard responses.
- Managing contacts:
  - Add a contact,
  - Remove a contact,
  - Modify contact information,
  - Search a contact.
- Using contact history information.
- Using the workbin features to store interactions:
  - Get the Workbin's queues and views,
  - Put an interaction in the Workbin.
- Using the System features to get options about the application used in the Configuration Layer.
- Handling open media interactions:
  - Reply to an open media interaction such as phone call,

  - Reply to an open media interaction such as e-mail,
  - Transfer an open media interaction.
- Handling chat interactions:
  - Have a chat conversation,
  - Transfer a chat interaction.

Benefits

The Microsoft .NET framework enables developers to use the same tools and skill to develop software for a variety of system, using variety of programming languages. It can minimize conflicts among applications by providing compatibility among otherwise incompatible software components.

The Genesys Desktop .NET Toolkit is a .NET-compliant product, and benefits from .NET Framework technology.

Platform Requirements

For development, you need access to the following:

- Microsoft Visual Studio .NET.
- A working Genesys Integration Server.
- The Genesys Desktop .NET Toolkit libraries:
  - `AgentDesktopToolkit.dll`,
  - `AILLibrary.dll`,
  - `AilServicesPropProtocol.dll`.

Note: The `dll` files are available in the `<installation_directory>` directory of the Genesys Desktop .NET Toolkit Multimedia Application Sample.

Design Features

Service-Oriented Architecture

The Genesys Desktop .NET Toolkit is based on Service-Oriented Architecture (SOA). SOA is a specific type of distributed system in which features are exposed with services.

When you are using the Genesys Desktop .NET Toolkit APIs, you are dealing with service interfaces that do not manage anything locally. Each service defines a specific feature of your distributed system. Data management and

actions are performed by GIS and you are concerned only with the interface descriptions.

Multithreaded

The Genesys Desktop .NET Toolkit is thread-safe and therefore can be run in multithreaded environments.

In particular, parallel threads can make calls to the same services' methods at the same time without encountering issues.

Synchronization

At start-up, the Genesys Desktop .NET Toolkit establishes a link with GIS which performs your client-application requests. The communication with this server-side application is synchronous.

Connecting to Your Genesys Environment

Connections to Genesys servers are maintained by GIS. The client-side application developed with the Genesys Desktop .NET Toolkit can be notified of servers' statuses, namely the loss of a connection.

GIS can maintain connections to multiple T-Servers.

GIS is designed to work in a single-tenant environment. It is possible to create a multi-tenant application, but all configuration layer objects that your application uses must be specified in the `Tenants` tab of the application, and these names must be unique.

For further information, refer to the *Genesys Interaction SDK 7.6 Java Deployment Guide.*

Framework Compatibility

The Genesys Desktop .NET Toolkit connects—through GIS—to the following Genesys servers of the Genesys Framework Suite:

- Configuration Server—the Configuration Server stores configuration information such as application parameters, or objects description such as DNs, places, or persons. The library core monitors the configuration server to update modifications. The library provides full integration with Genesys configuration layer objects such as Agent, Place, and DN.
- T-Server—the Telephony Server handles telephone requests and events by communicating with switches.

For voice-only mode, your application should connect with a Configuration Server, at least one T-Server, and optionally a Contact Server (included with the Internet Contact Solution).

Refer to the *Genesys Integration Server 7.6 Deployment Guide* for further details and to the *Genesys Supported Media Interfaces* document for information on supported switches.

Configuring Your Application in the Configuration Layer

In order to have your custom .NET Toolkit application work with the Genesys Framework as an agent desktop, or in another capacity, you need to provide the Configuration Layer with specific information. You do this through an Agent Interaction Server `Application` object that GIS uses, and with the options you set for that Agent Interaction Server. The following information indicates which sections and which options in those sections you need to set in that `Application` object.

multimedia Section

In the `multimedia` section on the `Options` tab of your Agent Interaction Server `Application` object, set the following options:

email-default-queue

Valid Values: <Any valid queue name>

Specifies the queue used for any new e-mail interaction created by the desktop (for example, a new e-mail out, a reply, or an invitation).

collaboration-mode

Valid Values: `pull, push`

Enables agents to use the collaboration feature to invite other agents to help answer a question from a contact.

email-trsf-ext-queue

Valid Values: <Any valid queue name>

Specifies the queue that the client application uses for an e-mail transfer request to an external resource.

email-outbound-queue

Valid Values: <Any valid queue name>

Specifies the default queue used to send an e-mail.

email-drafts-workbin

Valid Values: <Any valid workbin names (as defined in the Configuration Layer)>

Used to store outgoing e-mails as drafts in a workbin when an agent clicks the `Save & Close` button.

collaboration-workbin

Valid Values: <Any valid workbin names (as defined in the Configuration Layer)>

Used for the desktop collaboration feature. When inviting an agent in pull mode, the internal invitation is stored in the agent workbin.

openMedia-default-queue

Valid Values: <Any valid queue name>

Specifies the queue used for any new open-media interaction created by the desktop (for instance, an sms reply to an open media interaction).

openMedia-outbound-queue

Valid Values: <Any valid queue name>

Specifies the default queue used to send an open-media interaction.

preview-park-queue

Valid Values: <Any valid interaction queue name>

Allows an agent to transfer ownership of a proactive interaction to the transfer target when a proactive interaction voice call is transferred.

This permits the appropriate post processing to be applied after the target of the transfer releases.

default-from-address

Valid Values: <An email address that is part of the from addresses list>.

The `default-from-address` option defines the email address that will be used by default in outgoing email.

media

Valid Values: <Any valid media names (as defined in the Configuration Layer), separated by commas>

The `media` option defines the various media available to an agent at login.

Notes: The `media` option can be defined either in the `multimedia` section of your Agent Interaction Server `Application` object (along with the above options), or on a given agent's `Annex` tab.

If you choose not to set this option, by default the agent is logged in on e-mail and chat.

---

kworker Section

In the `kworker` section on the `Options` tab of your Agent Interaction Server `Application` object, set the following option:

easy-newcall

Valid Values: `true, false`

If set to `true`, the knowledge worker only needs to click the `New Call` button in order to establish the call.

outbound Section

enable-chain-75api

Valid Values: `true, false`

If set to `true`, enables the use of the new 7.5 API with the `OutboundChain` class.

Note: If the `enable-chain-75api` option is set to `true`, then the `GetOutboundRecordForOutboundChain()` method should be used to access the Outbound Record in the OutboundService component. If the option is set to `false`, then you should use of the `GetOutboundRecord()` method.

Genesys Multimedia Compatibility

The Genesys Desktop .NET Toolkit connects—through GIS—to Genesys Multimedia and provides full multimedia support for voice, e-mail, chat interactions, and open media interactions.

The connectivity concerns the following servers of Genesys Multimedia:

- Interaction Server—This server manages voice, e-mail, chat interaction, and open media interaction information with the Genesys Framework.
- Chat Server—This server manages chat interactions between agents and web visitors.
- Universal Contact Server (UCS)—This database server is used to retrieve e-mails, history, and contact information. It also allows for the manipulation of contact histories and of the standard response library. This server is optional for an application designed to run in voice-only configuration.

For e-mail and open media handling, GIS must connect with a Configuration Layer and a UCS and Interaction Server (both included with Multimedia).

For chat handling, GIS should connect with a Configuration Layer and a Chat Server, a UCS, and an Interaction Server (all three included with Multimedia).

Outbound Campaign Support

The Genesys Desktop .NET Toolkit connects—through GIS—to the Genesys Outbound Solution:

- Outbound Campaign Server—This server controls and organizes outbound campaigns.

For Outbound Campaign handling, GIS should connect with a Configuration Layer, an Outbound Campaign Server, and at least one T-Server.

Voice Callback Support

The Genesys Desktop .NET Toolkit connects—through GIS—to the Genesys Universal Callback Solution:

- Callback Server—This server controls and organizes callback records.

For voice callback handling, GIS should connect with a Configuration Layer, a Callback Server, and at least one T-Server.

API Overview

The Genesys Desktop .NET Toolkit allows you to access the `AgentDesktopToolkit.dll` assembly. You can use the APIs namespaces and services to create custom applications or to modify your existing applications.

Namespaces

The .NET Toolkit provides you with a set of GUI components associated with the `AgentDesktopToolkit.dll` assembly, which contains the following namespaces:

- `AgentDesktopToolkit`—general components.
- `AgentDesktopToolkit.Agents`—components for agent features.
- `AgentDesktopToolkit.Callbacks`—components for callback features.
- `AgentDesktopToolkit.Contacts`—components for contact features.
- `AgentDesktopToolkit.ExtendedControls`—extended controls, providing for components' enhancement.
- `AgentDesktopToolkit.Histories`—components for managing contacts' histories.
- `AgentDesktopToolkit.Interactions`—components for general interaction features.
- `AgentDesktopToolkit.Interactions.Chat`—components for chat features.
- `AgentDesktopToolkit.Interactions.Mail`—components for e-mail features.

- `AgentDesktopToolkit.Interactions.OpenMedia`—components for openmedia features.
- `AgentDesktopToolkit.Interactions.Voice`—components for voice features.
- `AgentDesktopToolkit.Outbounds`—components for processing outbound interactions.
- `AgentDesktopToolkit.Resources`—components for resource features.
- `AgentDesktopToolkit.SRL`—components for managing the Standard Response Library feature.
- `AgentDesktopToolkit.Workbins`—components for workbin management.

Underlying Services and Events

The .NET Toolkit components implement and share a set of services that interact with GIS.

The .NET Toolkit components hide the services and their inherent complexity:

- The components' features mask the services' requests.
- The components refresh when they receive services' events.

A .NET Toolkit component does not necessarily implement any services. It can be linked to other .NET Toolkit components implementing the required features, as shown in Figure 3 on page 24.

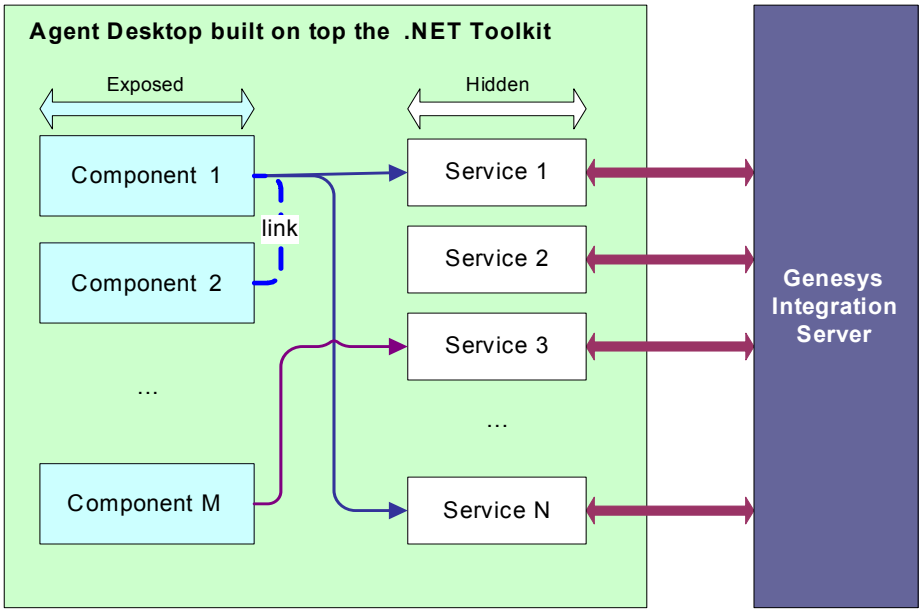

Figure 3: .NET Toolkit Components and Underlying Services

For further details about links, see "Working with Component Properties" on page 31.

Events received from the .NET Toolkit components could be received in another thread besides the GUI thread. For example, when the Agent component is instantiated, it has no association with a graphical container like a Form or a Control (in Windows.Forms). When the Agent component receives an (asynchronous) event, it is handled by an undetermined thread. Consequently, the following code is absolutely forbidden by Microsoft Windows:

```
agent1.AgentStatusChanged += new AgentDesktopToolkit.Agents.AgentAgentStatusChanged
                                                              (agent1_AgentStatusChanged);
...
private void agent1_AgentStatusChanged(AgentDesktopToolkit.Agents.AgentStatus status)
{
   MessageBox.Show("Hello World!"); // This is a Win32 action
}
```

The correct code should be:

```
agent1.AgentStatusChanged += new AgentDesktopToolkit.Agents.AgentAgentStatusChanged
                                                              (agent1_AgentStatusChanged);
...

private void agent1_AgentStatusChanged(AgentDesktopToolkit.Agents.AgentStatus status)
{
   if(InvokeRequired) //If true, we need to go into the GUI thread
      BeginInvoke(new AgentStatusChangedDelegate(AgentStatusChanged),
                                                                 new object[] {status});
    else
      AgentStatusChanged(status);
}

private delegate void AgentStatusChangedDelegate(AgentDesktopToolkit.Agents.
                                                                    AgentStatus status);

private void AgentStatusChanged(AgentDesktopToolkit.Agents.AgentStatus status)
{
   MessageBox.Show("//Hello World!//"); //This is a Win32 action
}
```

What's Next

The next chapter goes into greater detail about the examples provided with this SDK. It provides installation instructions and gives a basic explanation of the supported features.

Chapter

2 About the Examples

Source code examples in both Visual Basic .NET and C# exercise the use of the Rapid Application Development (RAD) GUI API and the Agent Interaction Services API features you are likely to use in your applications.

This chapter discusses them in the following topics:

Overview of the Code Examples

The following .NET examples demonstrate the use of important features of the .NET Toolkit API:

- The `ResourceService` example shows how to use .NET components on one side, and methods from the Agent Interaction Services on the other side.
- The `DatabaseLookupVoice` example is an application that handles voice interactions and retrieves information about contacts from a contact database, simulated by the `DatabaseLookup.xml` file.
- The `MultipleAttachedData` example is a voice application that enables the agent to manage attached data of several voice interactions.
- The `ActiveXCalendar` example is an application that manages an ActiveX component—the calendar—and voice interactions.
- The `ChangeLookAndFeelVoice` example is a simple application that manages voice interactions and shows how to customize the look and feel of .NET Toolkit components.

- The `RegisterEventSample` is an application that handles the registration/unregistration of workbin events, using the `RegisterEvent()` and `UnregisterEvent()` public methods from the Connection .NET Toolkit component.

For further information, see Chapter 3, ".NET Toolkit Examples," page 41.

Installing the Code Examples

Before installing the .NET Toolkit code examples, you need to install Microsoft .NET Visual Studio.

Location of the Code Examples

The source code examples are available by browsing the `\documentation\docs\ALL SDK Documentation.html` file located on your product CD. They are also available on the SDK Documentation CD.

Archive Content

Unzip the contents of the `76gd_exmpl_dotnet-toolkit.zip` archive to get the following directory tree:

```
76gd_exmpl_dotnet-toolkit/
   ResourceService/
      ResourceService/
      ExternalDependency/
   DatabaseLookupVoice/
      DatabaseLookupVoice/
      ExternalDependency/
   MultipleAttachedData/
      MultipleAttachedData/
      ExternalDependency/
   ActiveXCalendar/
      ActiveXCalendar/
      ExternalDependency/
   ChangeLookAndFeelVoice/
      ChangeLookAndFeelVoice/
      ExternalDependency/
   RegisterEventSample/
      RegisterEventSample/
      ExternalDependency/
```

Each example directory contains a sub-directory containing the Microsoft Visual Studio project files and an `ExternalDependency` directory containing the project references.

Using the Code Examples

Before you compile and run the .NET Toolkit examples, you must copy the required .NET Toolkit files into the `ExternalDependency/` directory:

`ail-configuration.xml`
`AgentDesktopToolkit.dll`
`AilServicesPropProtocol.dll`
`AilLibrary.dll`
`log4net.dll`
`C1spell.dll` (and dictionary files, such as, for instance, `C1SP_AE.dct`)

In the project's `Properties pages` dialog box, replace the `<ExternalDependency>` placeholders with the location of the `ExternalDependency` directory for the following properties:

- `Common Properties/References Path`
- `Configuration Properties/Debugging/Start Options/Working Directory`

Then, set the project properties working directory location for this project with the location of the `ExternalDependency` directory.

Then, set up proper configuration data in the `ExternalDependency/ail-configuration.xml` file. See "The XML Configuration File" on page 36 for further details.

To make GUI components available from the toolbox, you must add them to the toolbox, as follows:

- In the `View` menu, select `Toolbox`.
- Click right in the `Toolbox` window.
- Select `Add/Remove Items...`
- In the `Customize Toolbox` dialog box, browse the `AgentDesktopToolkit.dll` library.

Examples and Agent Interaction Services are designed to work with GIS. This server must be available on the network and expose the services. Also, an agent must be available: You must configure an `Agent` in your Configuration Layer before you run the examples.

.NET Toolkit Components

The .NET Toolkit components are easy to drag and drop into a form, allowing you to rapidly build an agent desktop. Although the .NET Toolkit components appear easy to use as independent objects, they have been designed to work in concert. Your development efforts will proceed more smoothly if you employ these components according to their designed logic.

Moreover, the namespaces forming the assembly also contain classes to facilitate the use of the .NET Toolkit components.

The following subsections should familiarize you with the .NET Toolkit components and help you to take advantage of the components.

Types of Components

In the `Toolbox`, you have a set of components that you can drag and drop into a form (or into a panel, and so on). There are two types of components:

- Those that have a visual representation in the form to which they have been added.
- Those that become associated with the form, but that are hidden and are only represented by a reference below the form itself.

Visible Components

Most of the components have a GUI container that automatically appears in the form. Figure 4 shows an `AgentDirectory` component, that is still selected in the `Toolbox`, and that has been dragged and dropped into the form. A table appears in the `Form1` window.

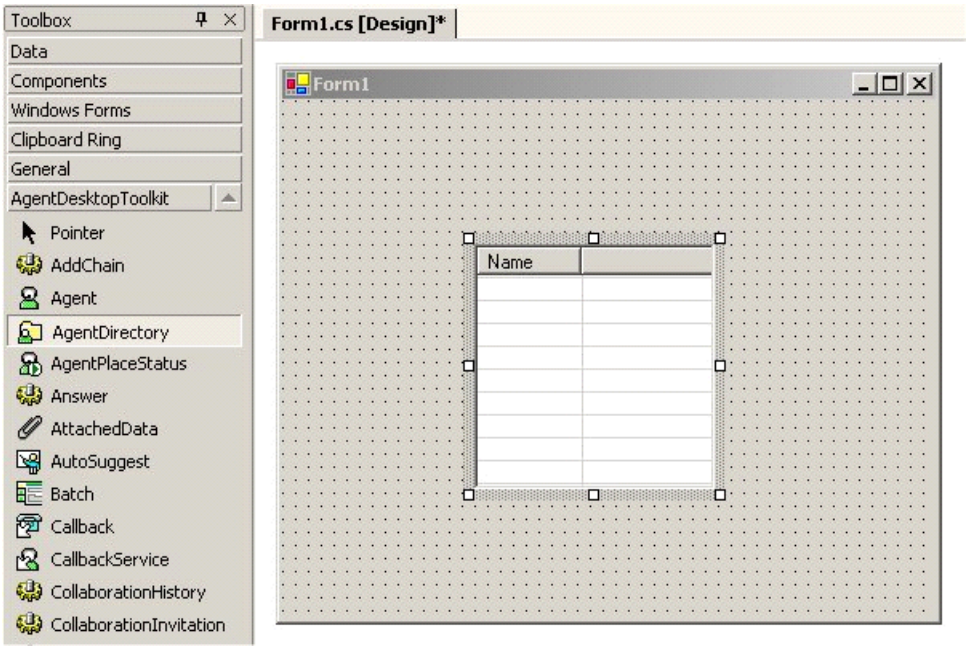

Figure 4: An AgentDirectory Component in a Form

Hidden Components

Some of these components have no GUI container appearing inside the form, and they appear at the bottom of the form window in the design sheet, as for example the `Connection` component shown in Figure 5 on page 31.

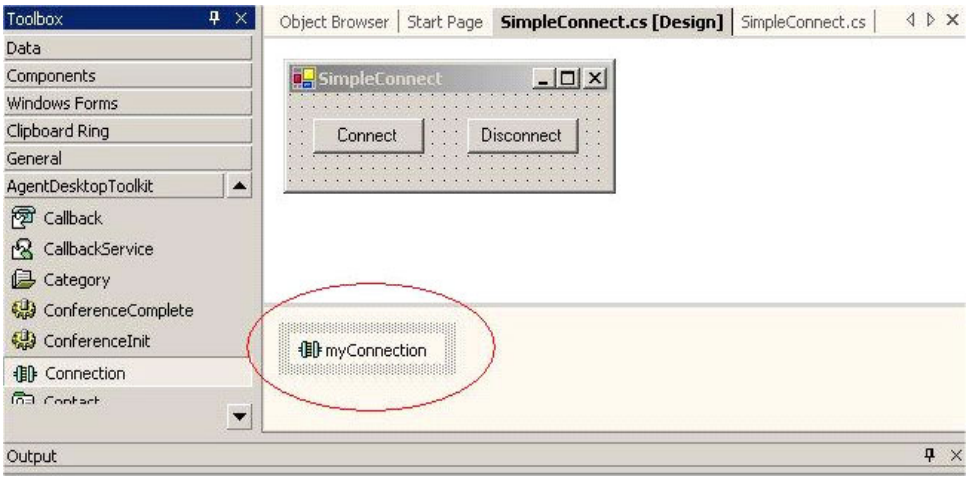

Figure 5: A Connection Component Dropped in a Form

Working with Component Properties

Once your application starts, you must initialize the .NET Toolkit components. This initialization activates the services and events managers for your GUI components and allows your application to interact with the Genesys Framework.

To initialize a component, you must set the value of some specific properties. To determine which properties are required for the initialization, look for the `Initialize()` method and properties of a given component in the *Genesys Desktop 7.6 .NET Toolkit API Reference*.

The .NET Toolkit components use two types of properties:

- "CTI Properties".
- "Links Between Components".

CTI Properties

CTI properties are common properties of the `Properties` windows and they include all information related to your application configuration in the Genesys Framework.

CTI properties are affected when your application changes its environment in one of the following ways:

- You have changed the location of your server and use new port values, or you have changed values in the Configuration Layer. Such a change, for instance, would affect the URL property of the `Connection` component, which is required to connect to the server-side application.
- Your component is updated regarding some CTI events. For example, the `Agent` component is updated with respect to the CTI values used by the person logging in.

The Agent component, for instance, relies on the following CTI properties:

- Agent login and password.
- Place of the agent.
- Workmode of the agent.

The Interaction component, on the other hand, relies on the following CTI properties:

- InteractionId.
- PhoneNumber.
- Extensions.

Links Between Components

The .NET Toolkit components have been designed to work together, and you have to follow some rules to obtain the expected behavior.

To work together, two components must be linked. Links are defined as <Type>Link or <Type>AutoLink properties, where <Type> is a component type. For example, the ConnectionLink property of a component refers to a Connection instance.

Linking Components

You can link components in two ways:

- The manual mode.
- The automatic mode.

In each mode, you can set up the components' link property in two ways:

- Graphically.
- By hand-coding.

Link Mode

In the manual mode, you must set the <Type>Link property, then call the Initialize() method for each component to be initialized.

In the automatic mode, you must set the <Type>AutoLink property of each component, and then you must initialize only the first component of the links' chain. All the other components automatically initialize.

Note: To avoid unpredictable results, do *not* mix manual and automatic modes when you set links.

Figure 6 on page 33 shows an Agent component that is auto-linked to the Connection component. When the Connection component initializes (by calling its connect() method), then the Agent component automatically initializes.

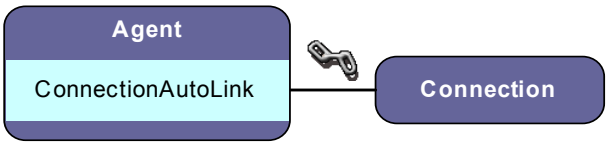

Figure 6: AutoLink Example

Graphical Linking

The first way to set up a component's link is to use the graphical interface. After dragging and dropping the components, select one of them. In the `Properties` window, set the `<Type>Link` or the `<Type>AutoLink` (according to the mode chosen) property from the available options in the drop-down list. See Figure 7 on page 33.

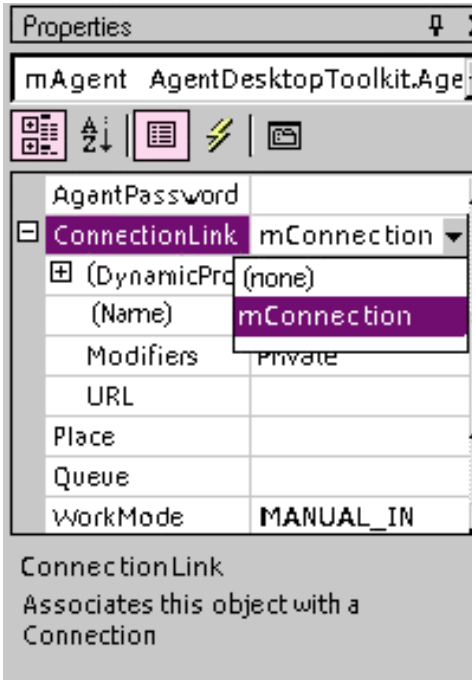

Figure 7: Available Connections for the Agent.ConnectionLink Property

In general, this approach is used to set logical and definitive links between the main components during the application build.

Hand-Coded Links

The second way to set the component link is to code it yourself. Once you have instantiated a component in your application, write the code for the link.

For example, for the manual mode, the corresponding code is:

```
mAgent.ConnectionLink = mConnection;
```

The above code snippet is equivalent to Figure 7 on page 33.

Warning! If, in your application code, you change the links of a component, then, you must reinitialize the component to take into account the new links.

Exclusive Links

A component can have more than one link, but some links are exclusive. This means that, if you activate an exclusive link, you cannot activate some of the other links. Links are exclusive for logical reasons. For example, the `Notepad` component displays notes attached to an interaction. The `Notepad` component might have the following two links:

- `InteractionAutoLink`: If this property has a reference, the `Notepad` displays the notepad of the associated interaction.
- `HistoryAutoLink`: If this property has a reference, the `Notepad` displays the notepad of the interaction selected in the `History` component.
- `InteractionId`: If this property contains an interaction ID, the `Notepad` displays the notepad of the interaction corresponding to the ID.

The `Notepad` component, however, cannot display the notepad of two distinct interactions at the same time. Therefore, only one link is available at a given time.

Another reason for link exclusivity might be link inheritance. For example, if the `InteractionVoice` component has a value set for a `BatchAutoLink` property or for a `ConnectionLink`, it is an exclusive link by inheritance. Figure 8 displays the inherited links for an `Interaction` linked with a `Batch`.

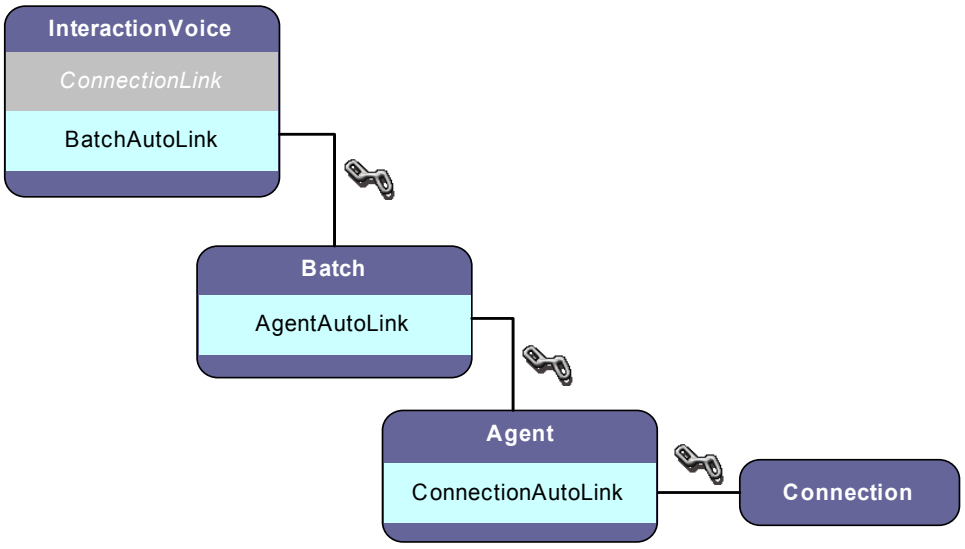

Figure 8: Links Examples for an Interaction Component

Figure 8 shows an `InteractionVoice` component having an inherited link from a `Connection` component, if the `InteractionVoice` uses its `BatchAutoLink` property. Therefore, the `BatchAutoLink` property prevents the `InteractionVoice`

component from implementing its `ConnectionLink` property as well—it cannot handle two concurrent `Connection` links.

Note: To identify exclusive links, see the `<Component>.initialize()` method description in the *Genesys Desktop .NET Toolkit API Reference*.

Delegates

Each component has a related set of delegates that helps in fine-tuning your application. There are two types of delegates:

- GUI delegates for GUI events, such as mouse clicks, mouse positions, element selected, and so on.
- Component-specific delegates for component changes such as interaction status, DN status, record status, and so on.

Delegates have explicit names, for example:

```
delegate <component_name><event_type>(<argument_list>);
```

Connection and Agent Components

The Genesys Desktop .NET Toolkit allows you to build client agent-desktop applications. As such, the `Agent` and `Connection` components are mandatory for your application development:

- The `Connection` component establishes a connection with GIS.
- The `Agent` component allows you to perform agent-related actions on media such as `login`, `ready`, `not ready`, and `logout`.

Connection

The `Connection` component does not have a GUI interface. To connect, call the `Connection.Connect()` method. This method takes into account the `Connection.EventPolling` and `Connection.URL` properties.

The `Connection.EventPolling` property is `true` by default and indicates whether your application pulls events or not. If you set the `Connection.EventPolling` property to `false`, the `Connection` component uses the notification mode for getting events.

The `Connection.URL` property depends on the protocol that your application uses to communicate with GIS:

- For GSAP, the value is `prop://[Server address]:[Server port]`.
- For SOAP, the value is `http://[Server Address]:[Server Port]/gis`.

If no `Connection.URL` property is defined, the application takes into account the `ail-configuration.xml` file. See “The XML Configuration File” on page 36.

The `Connection.AilServiceFactory` property accesses the underlying services. See "ResourceService" on page 42.

For further details about services, refer to the *Agent Interaction SDK 7.6 Services Developer's Guide*.

Agent

To initialize the `Agent` component, first set the `AgentId` and `Place` properties, then either:

- Set the `ConnectionLink` property and call the `Initialize()` method.
- Set the `ConnectionAutoLink` property.

The `Agent` component allows agent actions such as `login`, `ready`, `not ready`, and `logout`. To perform these actions, it takes into account the following properties (depending on the media related to the action):

- `AgentId`
- `AgentLogin`
- `AgentPassword`
- `Place`
- `Queue`
- `Workmode`

Refer to the API Reference for further details about these properties.

The XML Configuration File

When you start your .NET Toolkit application, it reads the `ail-configuration.xml` file to determine which protocols and options should be used for instantiating connection to GIS.

The XML configuration file is composed of two mandatory attributes that define the two different protocols that your application can use to connect the client. You can also define several optional attributes attached to these mandatory attributes.

Mandatory Attributes

In your XML configuration file, you must specify for the `factory` tag one of the following two attributes with their `url` option, according to the protocol used to communicate with GIS:

- For GSAP:
  - `PropFactory`—The factory name.
  - `url` option—The value is `prop://[Server address]:[Server port]"`.
- For SOAP:

- `WebServicesFactory`—The factory name.
- `url` option—The value is `http://[Server Address]:[Server Port]/gis`.

Optional Attributes

Table 1 on page 37 shows all the attributes that you can define for GSAP.

Table 1: Optional GSAP Attributes

Name	Type	Description
`logger`	string	The path to the log file.
`logger.level`	string	The level of the `ROOT` logger.
`logger.levels`	string	The levels of the loggers.
`initial.connect.timeout`	string	The timeout interval for the first connection to the GSAP Connector in synchronous mode.
`timeout.ack`	string	The timeout interval for acknowledgements from the server, in milliseconds.
`timeout.response`	string	The timeout interval for responses from the server, in milliseconds.
`timeout.check_interval`	string	The period for checking for the timeouts, in milliseconds.
`threads.max.worker`	string	Maximum number of threads in system pool. Should be greater than `50`.
`threads.max.io`	string	Maximum number of threads for IO operations in system pool. Should be greater than `50`.
`connector.buffersize.receive`	string	Receive buffer size for the sockets operations, in bytes. Should be greater than `8000` bytes.
`connector.buffersize.send`	string	Send buffer size for the sockets operations, in bytes. Should be greater than `8000` bytes.
`connector.tcpnodelay`	string	Should be set to `true`. Do not change this option.

Table 2 on page 38 shows all the attributes that you can define for SOAP protocol. For further information, see “XML Configuration File Example” on page 39.

Table 2: Optional SOAP Attributes

Name	Type	Description
`timeout`	int	The timeout interval for an XML web service client that waits for a synchronous XML web service request, to complete, in milliseconds. The default value is `100000` milliseconds.
`gis.checkSessionInterval`	int	The check session interval in seconds. The value `0` means no check is done.
`gis.username`	string	The GIS user name to log in the factory. Refer to Configuration Layer documentation for further details.
`gis.password`	string	The GIS password to log in the factory. Refer to Configuration Layer documentation for further details.
`gis.tenant`	string	The GIS tenant to use with the factory. Refer to Configuration Layer documentation for further details.
`gis.sessionId`	string	The GIS session identity to use with the factory. If you use this option, do not use `gis.username`, `gis.password`, and `gis.tenant`.
`notification.HTTPport`	int	The notification HTTP port. The default value is `0` (in which case, it is the remote system that chooses an open port on your behalf).
`notification.createHTTPchannel`	bool	Specifies the creation, or not, of an HTTP channel. The default value is `true`.
`notification.objectURI`	string	Specifies the remote object URI (Universal Resource Identifier). By default, the URI is generated by the `WebServiceFactory`.
`notification.reachableURL`	string	Indicates it is a reachable URI from the server.

Table 2: Optional SOAP Attributes (Continued)

Name	Type	Description
service-point-manager.defaultConnectionLimit	int	The service point manager's connection limit. The default value is 2.
service-point-manager.maxServicePointIdleTime	int	The service point manager's maximum idle time. The default value is 900,000 milliseconds (15 minutes).

XML Configuration File Example

The following is an example of an XML configuration file for a Genesys Desktop .NET Toolkit application using GSAP to communicate with GIS:

```
<?xml version="1.0"?>
<configuration default-factory="PropFactory"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<factory name="PropFactory"
         classname="com.genesyslab.ail.propprotocol.PropFactory"
         assembly="AilServicesPropProtocol">

   <option name="url"
           type="string"
           value="prop://[Server address]:[Server port]"/>

   <option name="logger" type="string" value="log.log"/>

   <option name="logger.level" type="string" value="DEBUG"/>

   <option name="logger.levels"
           type="string"
           value="BinaryFormatter:FATAL"/>

   <option name="timeout.ack" type="string" value="20000"/>

   <option name="timeout.response" type="string" value="60000"/>

   <option name="timeout.check_interval"
           type="string"
           value="1000"/>

   <option name="threads.max.worker" type="string" value="100"/>

   <option name="threads.max.io" type="string" value="100"/>
```

```
   <option name="connector.buffersize.receive"
           type="string"
           value="524288"/>

   <option name="connector.buffersize.send"
           type="string"
           value="524288"/>

   <option name="connector.tcpnodelay"
           type="string"
           value="true"/>
</factory>

<factory name="WebServicesFactory"
       classname="com.genesyslab.ail.WebServicesFactory"
       assembly="AilLibrary">
   <option name="Url"
           value="http://[Server Address]:[Server Port]/gis" />
   <option name="gis.username" value="default" />
   <option name="gis.password" value="password" />
<!--- OPTIONAL
   <option name="gis.sessionId" value="1234567"/>
   <option name="notification.HTTPport" type="int" value="10000"/>
   <option name="notification.createHTTPchannel"
           type="bool"
           value="true"/>
   <option name="notification.objectURI" value="NotifLoad"/>
   <option name="gis.checkSessionInterval" type="int" value="900"/>
   <option name="service-point-manager.defaultConnectionLimit"
           type="int"
           value="10"/>
   <option name="notification.reachableURL"
           value="http://localhost:8080/gis"/>
   <option name="service-point-manager.maxServicePointIdleTime"
           type="int"
           value="90000"/>
END OPTIONAL -->
</factory>
</configuration>
```

Chapter

3 .NET Toolkit Examples

This chapter is intended to give a consistent explanation of the code examples delivered with this developer's guide. It contains the following sections:

Prerequisites

To follow the discussion in this chapter, you will need:

- The *Genesys Desktop 7.6 .NET Toolkit API Reference*, available on both Product and Documentation CDs.
- The source code for the code examples.

See Chapter 2 on page 27 for further information about installing the code examples.

Each `<example name>` directory has two directory structures:

- `<example name>` contains the MS Visual Studio project files:
  - `<example name>.csproj`—The `<example name>` project file.
  - `Form<example name>.cs`—The example source file.
- `ExternalDependency` contains all the references you need. To implement this example, copy the following files into this directory:
  - `ail-configuration.xml`—The XML configuration file.
  - The Genesys Desktop 7.6 .NET Toolkit `dll`s.

References to `dll` files and other dependencies already exist in the project and are available if the `ExternalDependency` directory contains the correct files.

At application build time, the IDE copies the referenced `dll` files into the build directories (`debug/release`).

Note: At the project's first build, Microsoft Visual Studio performs all the required `dll` builds for the ActiveX components.

For other details, refer to the `Readme.html` file delivered with the examples.

Three Steps to a .NET Toolkit Application

Now that you have been introduced to the Genesys Desktop .NET Toolkit, it is time to outline the steps you will need to take work with its events and objects. There are five basic things you will need to do in your .NET Toolkit applications:

- **Add .NET Toolkit components** to a form and set up properties.
- **Connect to GIS**. Most of the examples use a .NET Toolkit dialog box—`LoginForm`—to get the CTI information required to connect using the `Connection` component.
- **Implement delegates** to the appropriate .NET Toolkit components.

The new examples have been designed to make these steps stand out so that you can quickly learn to write your own real-world applications. Now it is time to see how they are implemented in the `ResourceService` example.

ResourceService

The `ResourceService` example is a .NET Toolkit application that uses one of the underlying Agent Interaction Services—the `IResourceService` interface—to see a summary of each DN's status.

This example enables the agent to log in and select a DN resource type. After making his or her choice, the agent validates it by clicking the `Get Resource Type` button. A list of DNs appears in a treeview. For each DN, the tree displays the corresponding summary, as shown in Figure 9 on page 43.

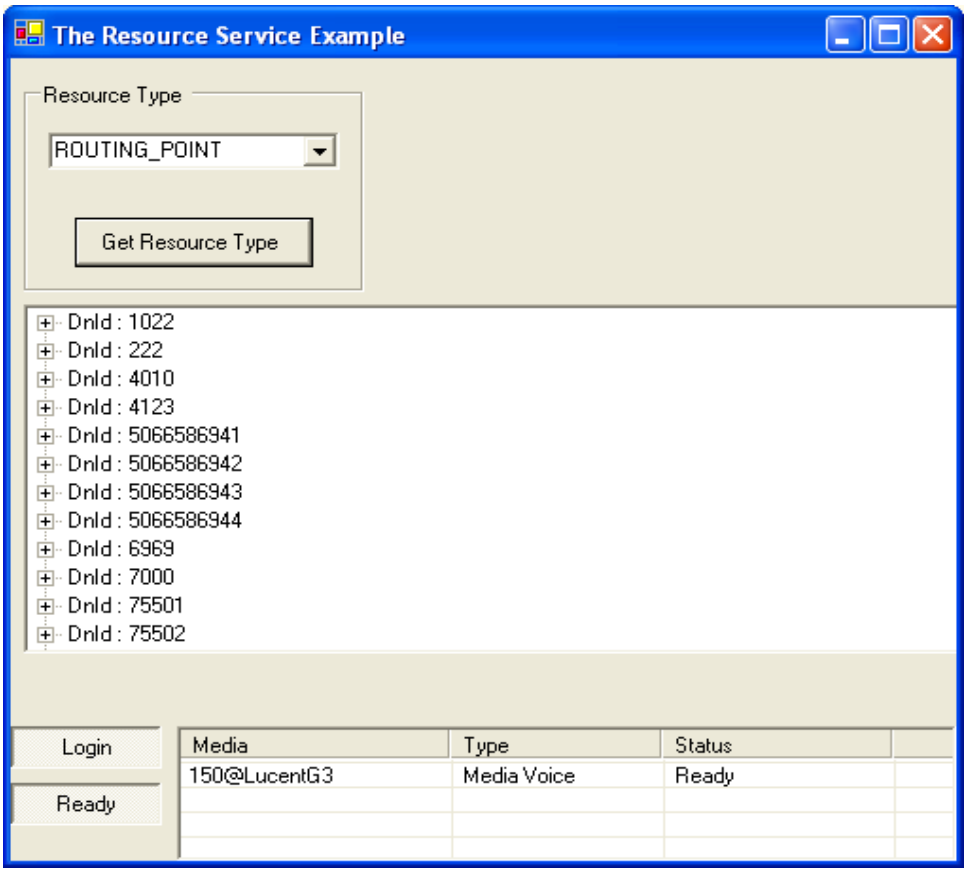

Figure 9: The Resource Service Example

Add .NET Toolkit Components

The `FormResourceService` form includes the following .NET Toolkit components:

- A `Connection` component—`mConnection`—that enables connection to GIS.
- An `Agent` component—`mAgent`—that logs the agent in.
- An `AgentPlaceStatus` component—`mAgentPlaceStatus`—that displays the agent's status on his or her place.

For this example, the values set to .NET Toolkit components' properties only concern automatic links. Figure 10 on page 44 shows these links, that can be set in the `Design` view of the project in Microsoft Visual Studio.

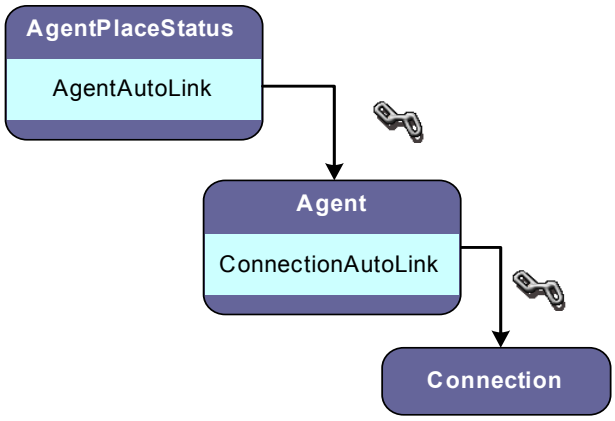

Figure 10: Links Set for the Resource Service Example

Except the `Connection` component, all the .NET Toolkit components have their `AutoLink` property set. This means that these components are initialized when the `FormResourceService()` constructor calls the `Connection.connect()` method.

For further information about links, see "Links Between Components" on page 32.

Connect to GIS

Before attempting a connection, the Resource Service example uses a dialog box to collect the CTI data required, first to connect, then to log in.

In this purpose, the `FormResourceService()` constructor creates and shows a `LoginForm` component, as shown here.

```
LoginForm loginForm = new LoginForm();
DialogResult nResult = loginForm.ShowDialog();
```

When the user clicks this dialog's `OK` button, `loginForm` contains the required values to fill in the `mConnection` and `mAgent`'s CTI properties, as shown here:

```
mConnection.URL = loginForm.URL;
mAgent.AgentId = loginForm.AgentId;
mAgent.AgentLogin = loginForm.AgentLogin;
mAgent.AgentPassword = loginForm.AgentPassword;
mAgent.Queue = loginForm.mQueue;
mAgent.Place = loginForm.mPlace;
```

At this point, the `mConnection` and `mAgent` components are ready to attempt a connection and then a login, as shown in the following code snippet:

```
try
{
   //Connect to GIS and intialize all auto-linked components
```

```
      mConnection.Connect();

      //Log in the Agent
      AgentDesktopToolkit.Agents.AgentStatus status = mAgent.Login();

      //If the login failed, disconnect from GIS
      if(status == null)
      {
         mAgent.Release();
         mConnection.Disconnect();
      }
   }
   catch(Exception e)
   {
      MessageBox.Show(e.ToString());
   }
   //...
```

Note: If an agent has a DN(s) in his Place, the `Agent.Login()` and the `Agent.Logout()` methods can sometimes return an incorrect `AgentStatus` due to the quick state change of the DN(s). You should not use possibles or status returned by those methods, instead use events to determine the correct status of the agent and DN(s).

Implement Delegates

The Resource Service example does not implement .NET Toolkit components' delegates. In this example, the implemented delegates concern the use of the `IResourceService` interface.

The `FormResourceService_Load()` method implements the `Form.Load` delegate, to be called at startup before the GUI form is visible for the user. This delegate gets the Agent Interaction Service's factory available through the `Connection` component, and creates a `IResourceService` instance, as shown in the following code snippet:

```
//Create Resource Service
mResourceService = mConnection.AilServiceFactory.createService(
                     typeof(IResourceService), null) as IResourceService;
```

The `btGetResourceType_Click()` method implements the `Button.Click` delegate of the `GetResourceInfo` button. The `btGetResourceType_Click()` method retrieves DN summaries through the `IResourceService` interface, as shown in the following code snippet.

```
// DnSummaryType to be retrieved: the one selected in the ComboBox
DnSummaryType type = (DnSummaryType) this.cbResourceType.SelectedItem;

//
DnSummaryDTO[] summaryDTO = mResourceService.getDnSummariesDTO( "", //any switch
                            type, // Type of concerned DNs
                            0,
                            -1,
                            new string[] {"*"});//all attribute values to be retrieved
```

Then, it displays the DN summaries into the tree view of the form. For further details about sevices, refers to the Agent Interaction Service 7.6 Documentation.

DatabaseLookupVoice

The `DatabaseLookupVoice` example is an application that handles voice interactions and retrieves information about contacts from a contact database, simulated by the `DatabaseLookup.xml` file.

When the agent selects an interaction in the `Batch` component, the example uses the contact ID available in the interaction's attached data to retrieve contact information from the file. Then, the example displays these information in a list view, as shown in Figure 11 on page 47.

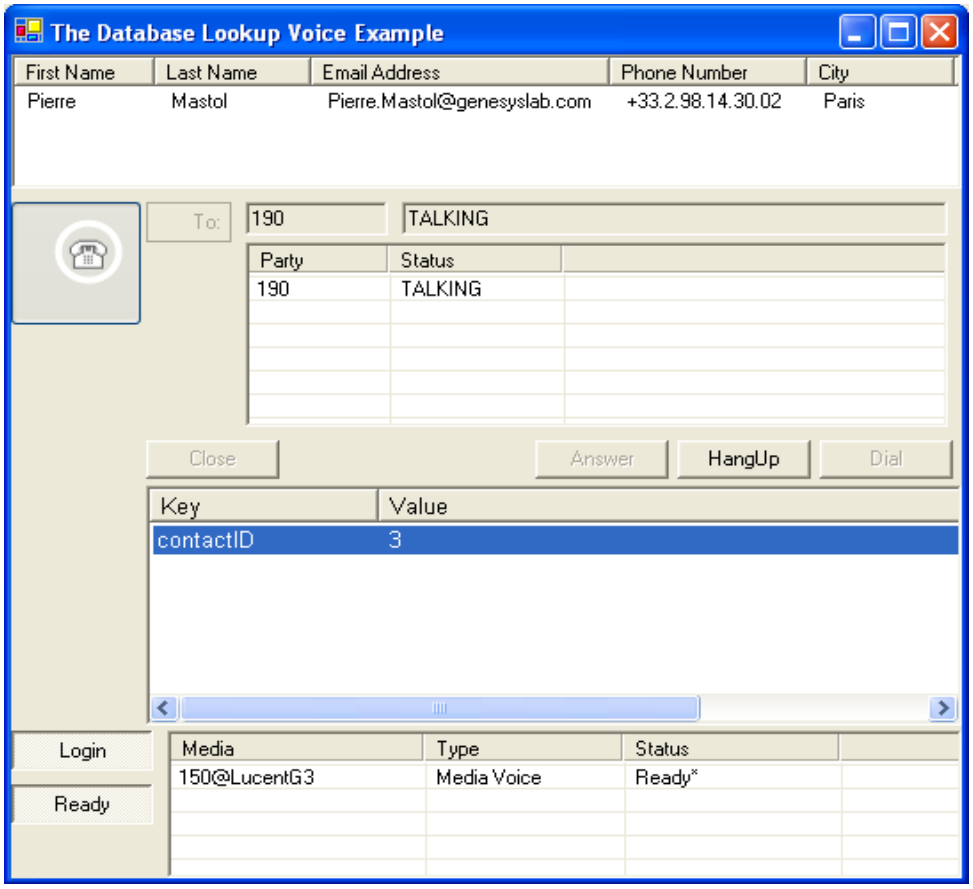

Figure 11: The Database Lookup Voice Example

Note: If the interaction has no contact ID specified in attached data, the example generates a contact ID for this interaction.

Add .NET Toolkit Components

The example use attached data in conjunction with the analysis of an XML file to provide the agent with more information about the customer.

The `FormDataBaseLookupVoice` form includes the following .NET Toolkit components:

- A `Connection` component—`mConnection`.
- An `Agent` component—`mAgent`.
- A `Batch` component—`mBatch`—manages all the agent voice interactions.
- An `InteractionVoice` component—`mInteractionVoice`—displays the voice interaction selected in `mBtach`.
- An `AgentPlaceStatus` component—`mAgentPlaceStatus`.
- An `AttachedData` component—`mAttachedData`.

For this example, the values set to .NET Toolkit components' properties only concern automatic links.

Figure 12 on page 48 shows these links, that can be set in the `Design` view of the project in Microsoft Visual Studio.

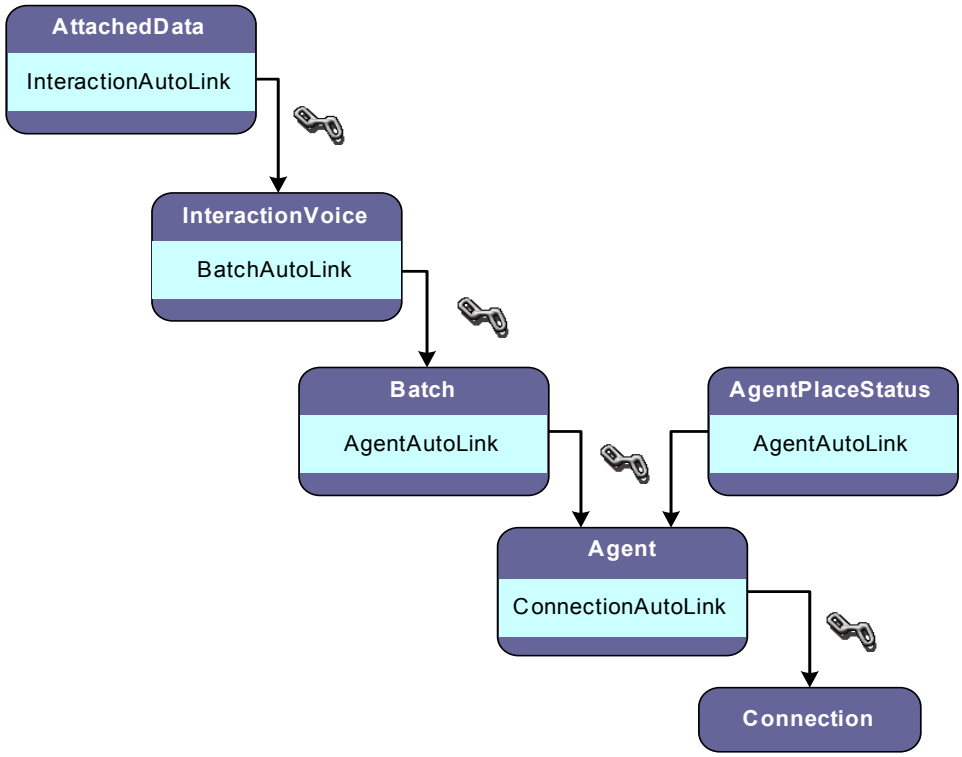

Figure 12: Links Set for the Database Lookup Voice Example

In this example, except the `Connection` component, the .NET Toolkit components have their `AutoLink` property set. This means that these components are initialized when the `FormDatabaseLookupVoice()` constructor calls the `Connection.connect()` method.

For further information about links, see "Links Between Components" on page 32.

Connect to GIS

The `DatabaseLookupVoice` example uses a dialog box to collect the CTI data required, first to connect, then to log in. In this purpose, the `FormDatabaseLookupVoice()` constructor uses a `LoginForm` component, then connects by calling the `Connection.connect()` method, as described for the Resource Service example, on page 44.

Implement Delegates

When the agent receives a voice interaction, this interaction contains attached data. The attached data's `key` is `ContactID` and the attached data's `value` is `1`, `2`, `3`, or `4`.

To simulate that the framework adds contact IDs to interactions, the `mAttachedData_Initialized()` method implements the `AttachedData.Initialized` delegate. This method checks whether or not the `mAttachedData` component includes a contact ID. In case there is no contact ID available, the method adds a random one to attached data, as shown here:

```
// Generates randomly a contact ID
// and adds it as the value of the "contactID" key
if(!this.mAttachedData.KeyValueData.Contains("contactID"))
{
   int randomNumber = randComp.Next(1,5);
   this.mAttachedData.Add("contactID",randomNumber.ToString());
}
```

When the agent selects a batch button, the autolinked components update accordingly to display the selected interaction. So, the example synchronizes the update of the listview—that displays contact information—with the selection of batch buttons.

In this purpose, the `mBatch_InteractionSelected()` method implements the `Batch.InteractionSelected` component. It retrieves the value of the `contactID` attached data and reads the corresponding information in the XML file `DatabaseLookup.xml`. Then, the example updates the list view with contact information.

See the following code snippet used for this process.

```
string contactIDStr = "";
//Search for the "contactID" key in attached data.
if (this.mAttachedData.KeyValueData.Contains("contactID"))
{
   //Retrieves from Attached Data the value that corresponds to the "contactID" key
   contactIDStr= this.mAttachedData.KeyValueData["contactID"].ToString();
   XmlDocumentdoc= new XmlDocument();
   //Loads the xml file that contains all contact descriptions
   doc.Load("DatabaseLookup.xml");
   //Searches for the "contactID" entry in the xml file.
   XmlNodeListcontactIDList = doc.SelectNodes("//TheAll/Code");
   for(int i=0; i< contactIDList.Count ; i++)
   {
      XmlNodecontactID = contactIDList[i];
      string id =contactID.SelectSingleNode("Id").InnerText;
      //If IDs match,retrieves the associated description
      if (id.CompareTo(contactIDStr) == 0)
      {
```

```
        //Retrieves the contact information
        string firstname = contactID.SelectSingleNode("FirstName").InnerText;
        string lastname= contactID.SelectSingleNode("LastName").InnerText;
        string phonenumber = contactID.SelectSingleNode("PhoneNumber").InnerText;
        string emailaddress= contactID.SelectSingleNode("EmailAddress").InnerText;
        string city= contactID.SelectSingleNode("City").InnerText;

        //Updates the list view with the contact information
        ListViewItem lvi = new ListViewItem(newstring[5]
   {firstname,lastname,emailaddress,phonenumber,city});
        this.lvContact.Items.Add(lvi);
        return;
      }
   }
}
```

MultipleAttachedData

The `MultipleAttachedData` example is a voice application that enables the agent to manage attached data of several voice interactions.

The `Interaction Attached Data` component displays attached data of the interaction selected in the batch. The `Connection Attached Data` component displays attached data of one of the interactions available through the application's connection, as shown in Figure 13 on page 51.

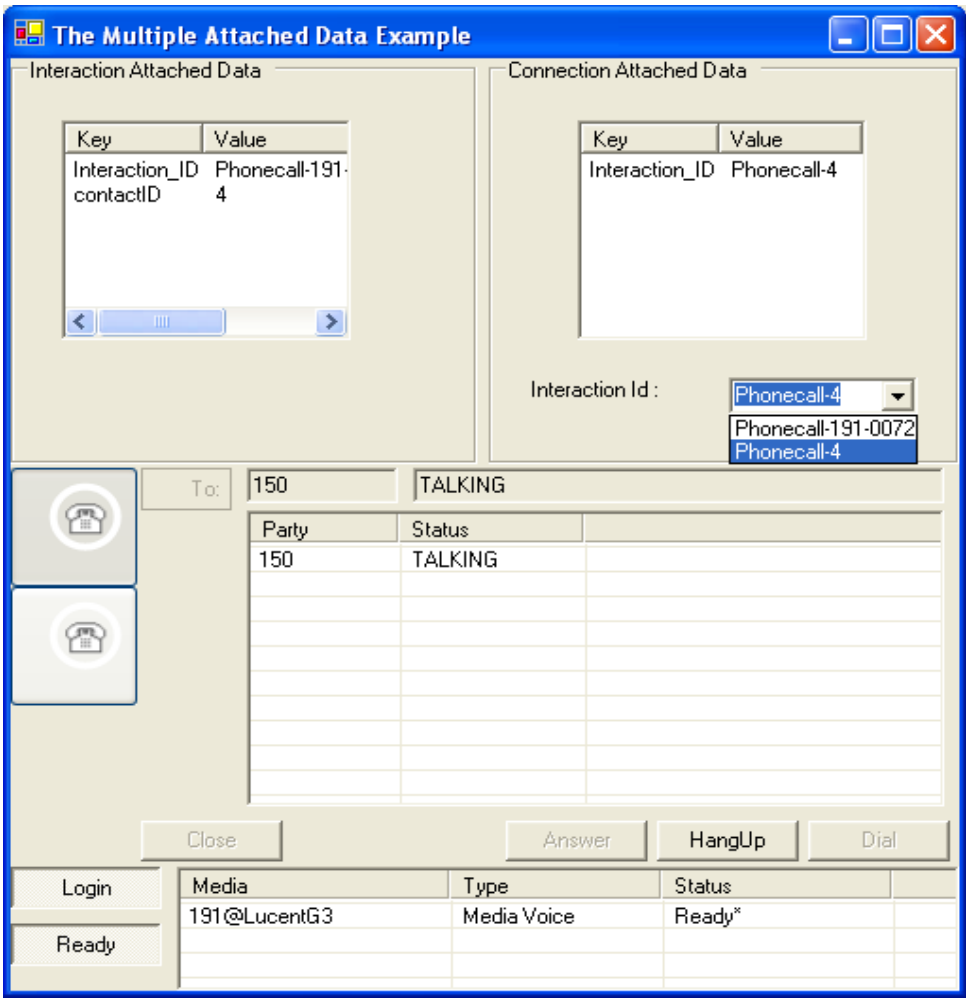

Figure 13: The Multiple Attached Data Example

In Figure 13, the agent selects the interaction identifier `Phonecall-1` in the combo box. The corresponding data appears in `Connection Attached Data`.

Add .NET Toolkit Components

The `FormMultipleAttachedData` form includes the following .NET Toolkit components:

- A `Connection` component—`mConnection`.
- An `Agent` component—`mAgent`.
- A `Batch` component—`mBatch`—manages all the agent voice interactions.
- An `InteractionVoice` component—`mInteractionVoice`—displays the voice interaction selected in `mBatch`.
- An `AgentPlaceStatus` component—`mAgentPlaceStatus`.

- Two `AttachedData` components:
  - `mADInteraction` is linked to the `InteractionVoice` component and is visible as `Interaction Attached Data` at runtime (see Figure 13 on page 51).
  - `mADConnection` is linked to the `Connection` component and is visible as `Connection Attached Data` at runtime (see Figure 13 on page 51). It requires the `InteractionId` property to determine which interaction's `attachedData` should be displayed.

Consequently, the example can display at the same time attached data of two different interactions.

As shown in Figure 14 on page 52, this example uses autolinks and simples links for the `mADConnection` attached data component.

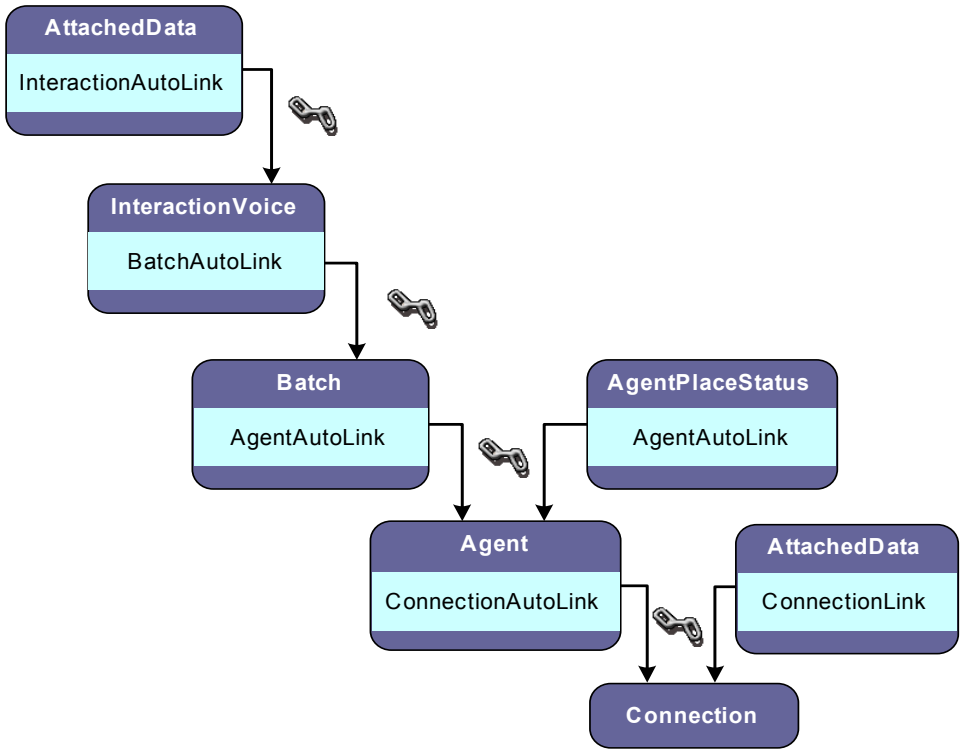

Figure 14: Links Set For the Multiple Attached Data Example

In this example, most of the .NET Toolkit components have their `AutoLink` property set. This means that these components are initialized when the `FormMultipleAttachedData()` constructor calls the `Connection.connect()` method.

The second `AttachedData` component—`mADConnection`—implements a manual link—`ConnectionLink`. So this example must manage the initialization of this component. See "Implement Delegates" on page 53.

For further information about links, see "Links Between Components" on page 32.

Connect to GIS

The `MultipleAttachedData` example uses a dialog box to collect the CTI data required, first to connect, then to log in. In this purpose, the `FormMultipleAttachedData()` constructor uses a `LoginForm` component, then connects by calling the `Connection.connect()` method, as described for the Resource Service example, on page 44.

Implement Delegates

The `MultipleAttachedData` example implements two delegates of the `Batch` component to handle the `cbInteractionsId` combo box—which lists the IDs of interactions available—and to synchronize with the `mADConnection` component—which displays attached data of the interaction selected in the combo box.

The `mBatch_InteractionSelected()` method implements the `Batch.InteractionSelected` delegate, to be called when the interaction is selected in the `batch` component. This method adds the ID of the selected interaction to the `cbInteractionsId` combo box if it does not already belong to the list displayed, as shown in the following code snippet.

```
if (!this.cbInteractionsId.Items.Contains(interactionId))
{
   this.cbInteractionsId.Items.Add(interactionId);
   this.mADInteraction.Add("Interaction_ID",interactionId);
}
```

The `mBatch_InteractionClosed()` method implements the `Batch.InteractionClosed` delegate, to be called when the interaction is closed. This method removes the interaction identifier from the list that the combo box displays. If the ID to be removed is the one selected in the list, the method releases the `mADConnection` component which displays attached data associated with this ID, as shown here:

```
if (this.cbInteractionsId.Text == interactionId)
{
   this.mADConnection.Release();
   this.cbInteractionsId.Text = "";
}
this.cbInteractionsId.Items.Remove(interactionId);
```

Finally, the `cbInteractionsId_SelectedIndexChanged()` method implements the delegate, to be called when the user selects an interaction in the combo box. It initializes the `mADConnection` component, which displays attached data associated with the selected interaction ID. The following code snippet illustrates this process.

```
if (this.cbInteractionsId.Text != "")
{
   this.mADConnection.InteractionId = this.cbInteractionsId.Text;
   this.mADConnection.ConnectionLink = this.mConnection;
   this.mADConnection.Initialize();
}
```

ActiveXCalendar

The `ActiveXCalendar` example is an application that manages an ActiveX component—the calendar—and voice interactions. It enables the agent to attach data (calendar dates) to voice interactions. It demonstrates an easy integration of an ActiveX component in an agent desktop application with a minimum of hand-coding.

Note: The only ActiveX components compatible with Genesys Desktop .NET Toolkit are Microsoft ActiveX Standard Components.

At runtime, when the agent handles a voice interaction, the ActiveX Calendar example displays the information of the current interaction (selected in the Batch) in the `InteractionVoice` and `AttachedData` components. If the agent selects a date in the ActiveX Calendar, the example adds this date into the interaction's attached data as shown in Figure 15 on page 55.

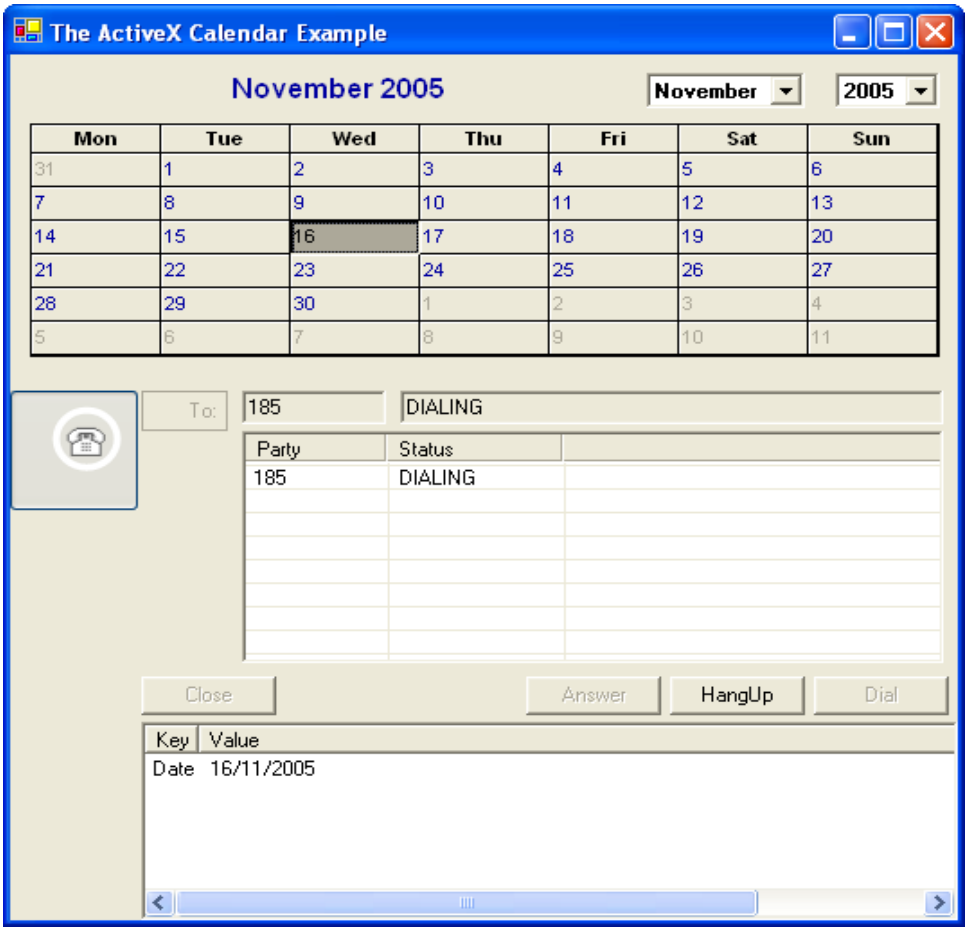

Figure 15: The ActiveX Calendar Example

Note that, at application startup, only the Microsoft Calendar ActiveX OCX is visible. The `InteractionVoice` and `AttachedData` components are visible only if an interaction is selected in the `Batch` component.

Add .NET Toolkit Components

The `FormActiveXCalendar` form includes the following .NET Toolkit components:

- A `Connection` component—`mConnection`—that enables connection to GIS.
- An `Agent` component—`mAgent`—that logs the agent in.
- A `Batch` component—`mBatch`—manages all the agent voice interactions.
- An `InteractionVoice` component—`mInteractionVoice`—displays the voice interaction selected in `mBtach`.
- An `AttachedData` component—`mAttachedData`—displays the attached data of the voice interaction displayed in `mInteractionVoice`.

The `FormActiveXCalendar` form also includes an `AxCalendar` component—`mCalendar`—that is the Microsoft ActiveX Calendar.

For this example, the values set to .NET Toolkit components' properties only concern automatic links.

Figure 16 shows these links, that can be set in the `Design` view of the project in Microsoft Visual Studio.

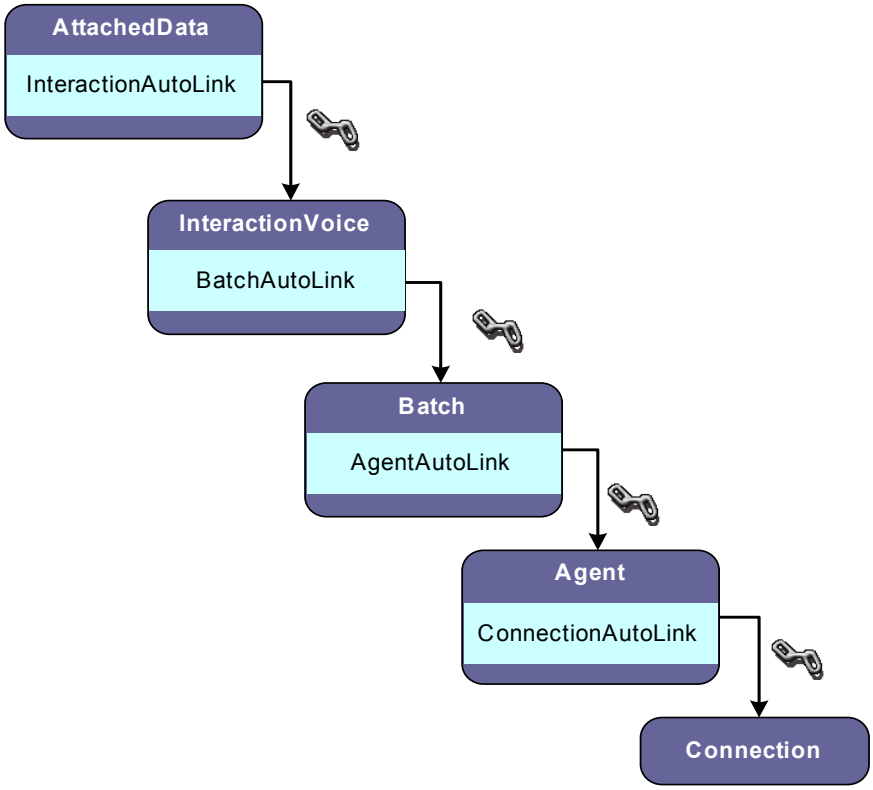

Figure 16: Links Set for the ActiveX Calendar Example

In this example, except the `Connection` component, the .NET Toolkit components have their `AutoLink` property set. This means that these components are initialized when the `FormActiveXCalendar()` constructor calls the `Connection.connect()` method.

For further information about links, see "Links Between Components" on page 32.

Connect to GIS

The ActiveX Calendar example uses a dialog box to collect the CTI data required, first to connect, then to log in. In this purpose, the `FormActiveXCalendar()` constructor uses a `LoginForm` component, then connects by calling the `Connection.connect()` method, as described for the Resource Service example, on page 44.

Then, the `FormActiveXCalendar()` constructor disables the `mInteractionVoice` and `mAttachedData` components because they should only be visible when a voice interaction is available for the agent.

```
//Hide the InteractionVoice and AttachedData components
this.mInteractionVoice.Visible = false;
this.mAttachedData.Visible = false;
```

Implement Delegate

The ActiveX Calendar example includes a `Batch` component to manage voice interactions and implements two of its delegates:

- `BatchInteractionCreated`:
  - The form calls this delegate when an interaction is created via the batch (the user created an interaction using the `Batch` or a ringing interaction is added to the `Batch`).
  - In this example, this delegate makes `mInteractionVoice` and `mAttachedData` components visible.

    ```
    this.mInteractionVoice.Visible = true;
    this.mAttachedData.Visible = true;
    ```

- `BatchInteractionClosed`:
  - This delegate is called when the selected interaction is closed.
  - In this example, this delegate makes `mInteractionVoice` and `mAttachedData` components invisible if there is no longer an interaction selected in the Batch.

    ```
    if (this.mBatch.InteractionIds.Length == 0)
    {
       this.mInteractionVoice.Release();
       this.mAttachedData.Release();
       this.mInteractionVoice.Visible = false;
       this.mAttachedData.Visible = false;
    }
    ```

When you open the project in the Design view, .NET Toolkit delegates are available in the CTI section of the `Events Properties` tool. The corresponding source code is available in the `FormActiveXCalendar.initializeComponent()` method, as shown in the following code snippet:

```
this.mBatch.InteractionCreated += new
   AgentDesktopToolkit.Interactions.BatchInteractionCreated(
      this.mBatch_InteractionCreated);
this.mBatch.InteractionClosed += new
   AgentDesktopToolkit.Interactions.BatchInteractionClosed(
      this.mBatch_InteractionClosed);
```

In this example, the ActiveX Calendar control has a minor task. Its purpose is to show that it is simple to interact with the .NET Toolkit components. Here,

when the user clicks in the Calendar, the `mCalendar_AfterUpdate` delegate adds the selected date to the `mAttachedData` component, as shown in the following code snippet:

```
private void mCalendar_AfterUpdate(object sender,
                                   System.EventArgs e)
{
   string str = mCalendar.Day + "/" + mCalendar.Month + "/"
                + mCalendar.Year;
   this.mAttachedData.Add("Date",str);
}
```

ChangeLookAndFeelVoice

The `ChangeLookAndFeelVoice` example is a simple application that manages voice interactions and shows how to customize the look and feel of .NET Toolkit components. It allows you to modify the background color, the foreground color, and the font of the agent's application.

The agent opens the `LookFeel` menu and selects the `BackColor` or `ForeColor` item. It opens the `Edit Colors` dialog box. Then, the agent selects a new color and clicks `OK`. The new setting is immediately taken into account, as shown in Figure 17 on page 59.

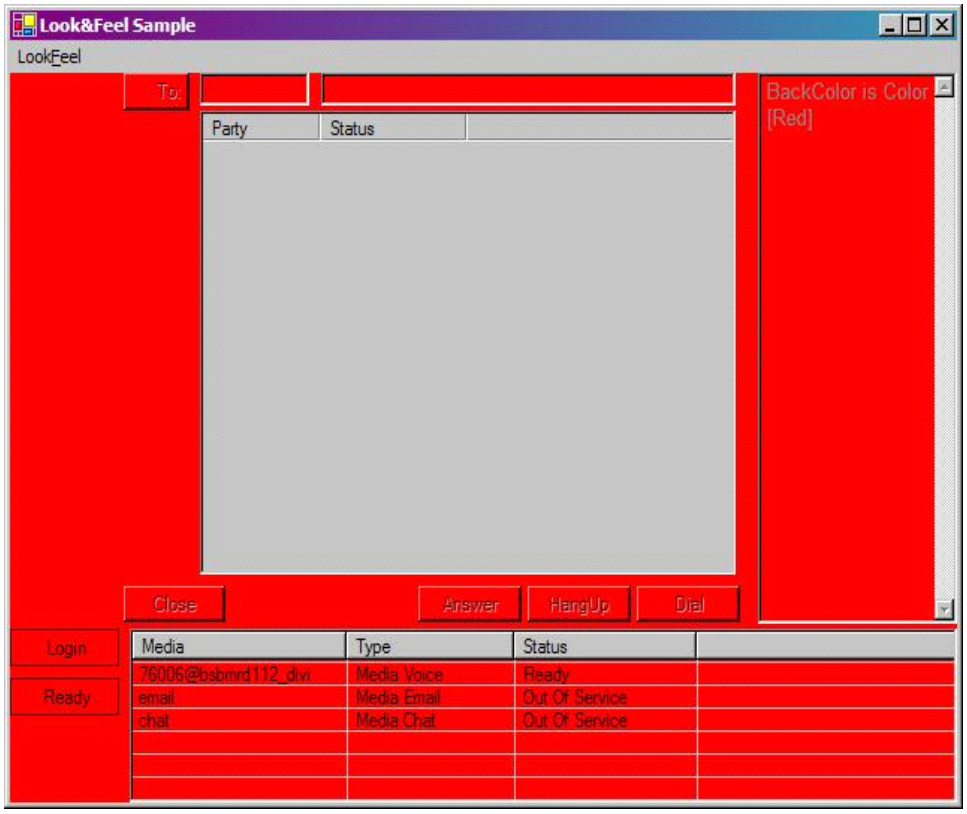

Figure 17: The Change Look and Feel Example

Add .NET Toolkit Components

The `FormChangeLookAndFeelVoice` form includes the following .NET Toolkit components:

- A `Connection` component—`mConnection`—that enables connection to GIS.
- An `Agent` component—`mAgent`—that logs the agent in.
- A `Batch` component—`mBatch`—manages all the agent voice interactions.
- An `InteractionVoice` component—`mInteractionVoice`—displays the voice interaction selected in `mBtach`.
- A `Notepad` component—`mNotepad`—displays the `Look and Feel` changes applied to the .NET Toolkit components.
- An `AgentPlaceStatus` component—`mAgentPlaceStatus`.

For this example, the values set to .NET Toolkit components' properties only concern automatic links.

Figure 18 on page 60 shows these links, that can be set in the `Design` view of the project in Microsoft Visual Studio.

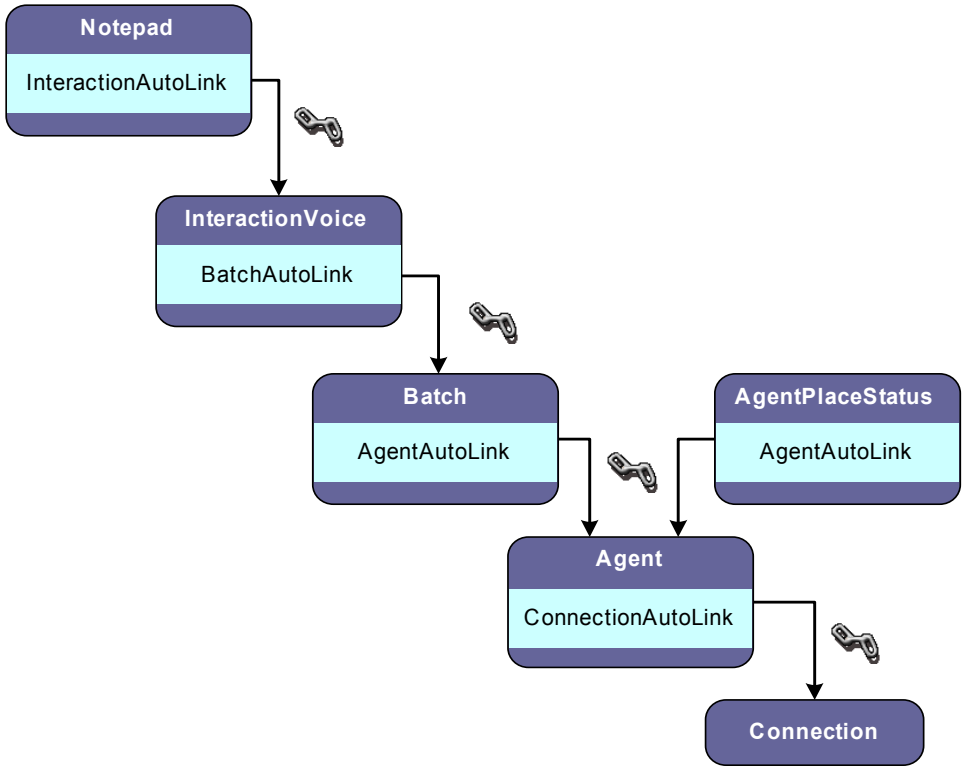

Figure 18: Links Set for the Change Look and Feel Example

In this example, except the `Connection` component, the .NET Toolkit components have their `AutoLink` property set. This means that these components are initialized when the `FormActiveXCalendar()` constructor calls the `Connection.connect()` method.

For further information about links, see "Links Between Components" on page 32.

Connect to GIS

The `ChangeLookAndFeelVoice` example uses a dialog box to collect the CTI data required, first to connect, then to log in. In this purpose, the `FormChangeLookAndFeelVoice()` constructor uses a `LoginForm` component, then connects by calling the `Connection.connect()` method, as described for the Resource Service example, on page 44.

Implement Delegates

The `ChangeLookAndFeelVoice` example does not implement .NET Toolkit components' delegates. In this example, the implemented delegates concern the GUI events on the menu provided to change font and colors of the application.

The `miBackColor_Click()` method implements the `MenuItem.Click` delegate, to be called when the agent clicks on the `BackColor` menu item. This delegate displays a dialog box to choose the color for the components' background. Then, the application applies the selected color to all the .NET components, as shown here:

```
if (this.colorDialog.ShowDialog() == DialogResult.OK)
{
   //Color selected in the color dialog box
   Color clr = this.colorDialog.Color;
   //Form
   this.BackColor = clr;
   //Batch
   this.mBatch.BackColor = clr;
foreach (AgentDesktopToolkit.Interactions.BatchButton bb in
this.mBatch.BatchButtons)
{
   bb.FlatStyle = System.Windows.Forms.FlatStyle.Standard;
   bb.BackColor = clr;
}
//AGENT PLACE STATUS
this.mAgentPlaceStatus.BackColor = clr;
this.mAgentPlaceStatus.ListViewAgentStatus.BackColor = clr;
setBackColorObject(this.mAgentPlaceStatus.ButtonLogin,clr);
setBackColorObject(this.mAgentPlaceStatus.ButtonReady,clr);
```

For some components, the `BackColor` property must be set for the component's subelements—for example, the `BackColor` property applies not only to the `AgentPlaceStatus` component, but also to its buttons and its `ListViewAgentStatus`. The example calls the `setBackColorObject()` method to modify colors of the sub-components, as shown for the `InteractionVoice` component.

```
//INTERACTION VOICE
this.mInteractionVoice.BackColor = clr;
this.mInteractionVoice.ListViewParty.BackColor = clr;
setBackColorObject(this.mInteractionVoice.ButtonAcceptAndDial,clr);
setBackColorObject(this.mInteractionVoice.ButtonAnswer,clr);
setBackColorObject(this.mInteractionVoice.ButtonClose,clr);
```

This example uses the same mechanism for the `ForeColor` and `Font` properties.

RegisterEvent

The `RegisterEvent` example is a .NET Toolkit application that registers/unregisters workbin events. The `OnConnectionEvent` method from `IConnectionNotification` interface is implemented to receive registered

events. Events are registered and unregistered using the `RegisterEvent` and `UnregisterEvent` public methods found in the `Connection` component.

By clicking on the Register Workbin Events button, all the events related to the agent's workbin will be displayed in the form's text window, as shown in Figure 19 on page 62. The Unregister Workbin Events button unregisters the workbin events and stops the display. To clear the text window, click on the Clear button.

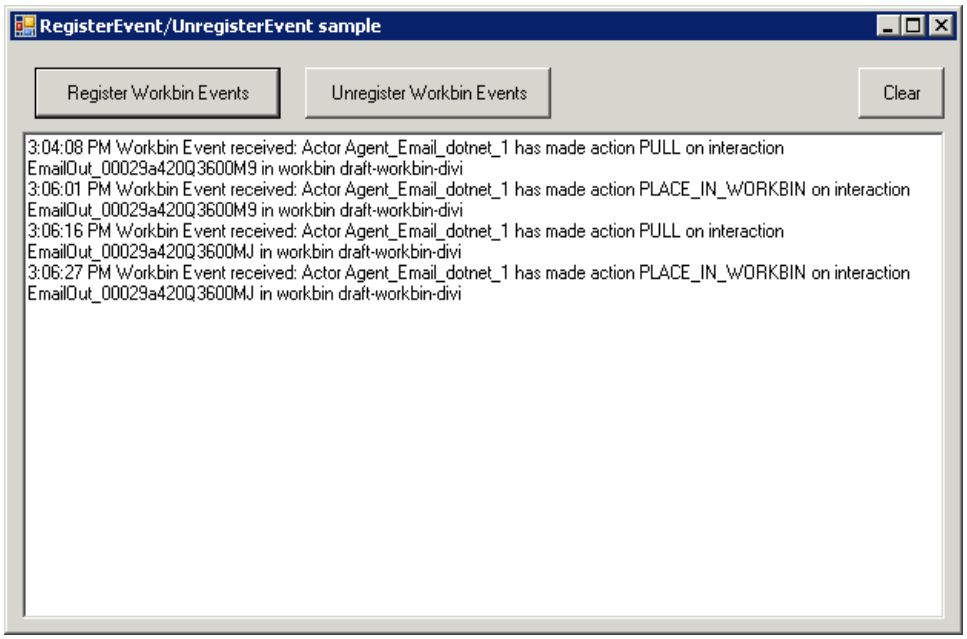

Figure 19: The Register Event Example

Add .NET Toolkit Components

The `MainForm` form includes the following .NET Toolkit components:

- A `Connection` component—`connection1`—that enables connection to GIS.
- An `Agent` component—`agent1`—that logs the agent in.

For this example, the values set to .NET Toolkit components' properties only concern automatic links. Figure 20 on page 62 shows these links, which can be set in the `Design` view of the project in Microsoft Visual Studio.

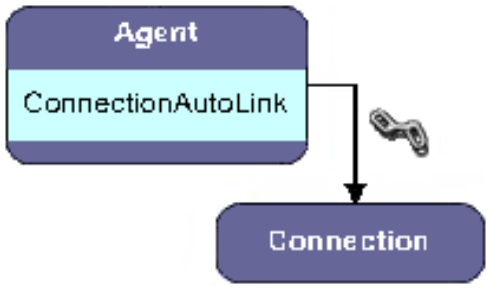

Figure 20: Links Set for the Register Event Example

In this example, the `Agent` component has its `AutoLink` property set. This means that the `Agent` component is initialized when the `MainForm()` constructor calls the `Agent.connect()` method.

For further information about links, see "Links Between Components" on page 32.

Connect to GIS

The `RegisterEvent` example uses a dialog box to collect the CTI data required, first to connect, then to log in. When the `MainForm()` loads, the `Agent` component is initialized and connects to GIS.

```
private void MainForm_Load(object sender, System.EventArgs e)
{
   agent1.ConnectionAutoLink = connection1;
   agent1.Connect();
   tbEvents.Clear();
}
```

Implement Delegates

The `RegisterEvent` sample implements one delegate, the `ConnectionEventDelegate`:

```
ConnectionEventDelegate(com.genesyslab.ail.ws._event.Event e);
```

The `OnConnectionEvent` method from the `IConnectionNotification` interface is implemented in order to receive registered events using the `RegisterEvent` public method of the `Connection` component. A delegate must be used to ensure that the graphical action will be made in the GUI thread:

```
void AgentDesktopToolkit.IConnectionNotification.OnConnectionEvent
                                                (com.genesyslab.ail.ws._event.Event e)
{
   if(InvokeRequired) //If true, we need to go into the GUI thread
   {
      Invoke(new ConnectionEventDelegate(this.ConnectionEvent), new object[] { e });
   }
   else
      ConnectionEvent(e);
}

private delegate void ConnectionEventDelegate(com.genesyslab.ail.ws._event.Event e);

private void ConnectionEvent(com.genesyslab.ail.ws._event.Event e)
{
   //We do some graphical stuff here....
}
```

Code Explanation

The `RegisterEvent` sample has three buttons:

- Register Workbin Events—Registers the events of all the workbins available to the agent.
- Unregister Workbin Events—Unregisters the events of all the workbins available to the agent.
- Clear—Clears the text box of all event information.

The `Click()` events of the Register Workbin Events and Unregister Workbin Events buttons contain the code that you will want to examine and utilize in your own custom build application.

Registering Events

An explanation of the code executed when the Register Workbin Events button is clicked follows:

```
private void btRegisterWBEvent_Click(object sender, System.EventArgs e)
{
   if(connection1 != null && agent1.Place != "")
   {
```

First, retrieve a list of workbins available for the agent.

```
      IWorkflowService mWorkflowService = connection1.AilServiceFactory.createService
                                    (typeof(IWorkflowService), null) as WorkflowService;
      WorkbinDTO[] workbins = mWorkflowService.getWorkbinsDTO(agent1.Place,null,
                                    new string[] {"workbin:id"},null);
      System.Collections.Specialized.StringCollection workbinsIdCollection =
                        new System.Collections.Specialized.StringCollection();
      if(workbins != null)
      {
         foreach(com.genesyslab.ail.ws.workflow.WorkbinDTO w in workbins)
         {
            foreach(KeyValue kv in w.data)
            {
               if(kv.key == "workbin:id")
                  workbinsIdCollection.Add(kv.value.ToString());
            }
         }
      }.....
```

Then, if at least one workbin exists for the agent, registration is made to WorkflowService/WorkbinEvent for all the workbins available to that agent. See `AilServices.chm` help file for additional information.

```
if (workbinsIdCollection.Count > 0)
{
   TopicsService [] topicServices = new TopicsService[1];
   topicServices[0] = new TopicsService();
   topicServices[0].serviceName = "WorkflowService";
   topicServices[0].topicsEvents = new TopicsEvent[1];
```

```
   topicServices[0].topicsEvents[0] = new TopicsEvent();
```

Here, the list of requested attributes is asked. The attributes are Ail Services "workbin:*" attributes.

```
   topicServices[0].topicsEvents[0].attributes = new string[]
      { "workbin:id","workbin:actorId","workbin:operationType","workbin:interactionId"};
   topicServices[0].topicsEvents[0].filters = null;
   topicServices[0].topicsEvents[0].eventName = "WorkbinEvent";
```

The trigger for this type of event uses a `KEY=WORKBIN` and a `VALUE=placeId:workbinId`.

```
   topicServices[0].topicsEvents[0].triggers = new Topic[workbinsIdCollection.Count];
   for(int index = 0; index < workbinsIdCollection.Count; index++)
   {
      topicServices[0].topicsEvents[0].triggers[index] = new Topic();
      topicServices[0].topicsEvents[0].triggers[index].key = "WORKBIN";
      topicServices[0].topicsEvents[0].triggers[index].value =
    agent1.Place.Replace(":","::") + ":" +
    workbinsIdCollection[index].Replace(":","::");
   }
```

Registration to these types of events uses the `RegisterEvent` public method from the `Connection` component.

```
   connection1.RegisterEvent(this,topicServices);
}
```

Displaying the Recieved Event

The `ConnectionEvent` method handles the incoming registered event and displays information about the event in the text box:

```
private void ConnectionEvent(com.genesyslab.ail.ws._event.Event e)
{
   string interactionId = "", actorId = "", workbinName = "",
                         operationType ="",workbinOperationType = "";

   if (e.attributes != null && e.attributes.Length > 0)
   {
      foreach(KeyValue kv in e.attributes)
      {
         if (kv.key == "workbin:id")
            workbinName = kv.value.ToString();
         if (kv.key == "workbin:actorId")
            actorId = kv.value.ToString();
         if (kv.key == "workbin:operationType")
            operationType = kv.value.ToString();
         if (kv.key == "workbin:workbinOperationType")
            workbinOperationType = kv.value.ToString();
         if (kv.key == "workbin:interactionId")
            interactionId = kv.value.ToString();
      }
```

```
      tbEvents.Text += string.Format("{0} Workbin Event received: Actor {1} has made
   action {2} on interaction {3} in workbin
   {4}\r\n",DateTime.Now.ToLongTimeString(),actorId,operationType,interactionId,workbin
   Name);
   }
}
```

Unregistering Events

To unregister from registered workbin events, the `UnregisterEvent` public method from the `Connection` component is used, as seen here:

```
private void btUnregisterWBEvent_Click(object sender, System.EventArgs e)
{
   connection1.UnRegisterEvent(this);
}
```

Index

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

X