Domino Designer 8: Web Agents and Web Services

Version 1.1
Copyright Information

©2008 wareSource.com

Part #DDWAWS8-1.1, updated for Domino Designer and Domino 8.0.1.

Under the copyright laws, this book may not be photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of wareSource.

While every reasonable precaution has been taken in the preparation of this book, the author assumes no responsibility for errors or omissions, nor for the uses made of the material contained herein and the decisions based on such use. No warrantees are made, express or implied, with regard to either the contents of this work, its merchantability, or fitness for a particular purpose. The author shall not be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the contents of this book.

In no event shall the author be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of business information, or any other loss) arising out the use of or inability to use this material, even if the author has been advised of the possibility of such damages.

Lotus, Domino, Domino Designer, ScreenCam, LotusScript, Notes/FX, Lotus Notes, Notes, iNotes, DataLens, Notes Minder, and Sametime are trademarks or registered trademarks Lotus Development Corporation and/or IBM Corporation. IBM, OS/2, AS/400, S/390, AIX, DB2, and WebSphere are registered trademarks of International Business Machines, Incorporated. Microsoft is a registered trademark and Windows, Visual Basic, and ActiveX are trademarks of Microsoft Corporation. Netscape and Netscape Navigator are trademarks of Netscape Communications Corporation. Java and JavaScript are trademarks of Sun Microsystems, Inc.

All other marks are the property of their respective owners.
# Table of Contents

- Topic 1: Introduction ........................................................................................................... 9
- Topic 2: WebQueryOpen Event Agents ............................................................................. 15
- Topic 3: WebQuerySave Event Agents .............................................................................. 35
- Topic 4: Web Agents ........................................................................................................ 43
- Topic 5: Web Agent Error Handling ............................................................................... 59
- Topic 6: Web Services Concepts .................................................................................... 65
- Topic 7: Web Service Properties .................................................................................... 93
- Topic 8: Web Service Programmer’s Pane ................................................................. 113
- Topic 9: Consume Web Services .................................................................................. 127
- Topic 10: Web Service Error Handling ....................................................................... 161
- Topic 11: Remote Debugging, Logging, and Profiling ............................................... 171
- Index ............................................................................................................................ 185
Description

The Domino HTTP server task is extremely versatile. In addition to its HTML rendering tasks, it also executes several types of design elements coded in Java or LotusScript:

- Agents run by a Form’s WebQueryOpen and WebQuerySave events.
- Agents run by the `?OpenAgent` URL command.
- Web Services that run in response to a SOAP request message.

This course covers the development and security of these design elements. You will also learn debugging and run time error handling techniques.

This course also takes a close look at Web Service client requirements and teaches you how to code a simple Web Service client using a SOAP toolkit.

Course goals

This course will teach you how to:

- Describe the differences and application of Agents run from a Form’s WebQueryOpen and WebQuerySave events and those run from the `?OpenAgent` URL command.
- Code Agents run by the Form’s WebQueryOpen event to work within the document context as well as perform lookups to other databases.
- Conditionally run WebQueryOpen Agents depending on document state.
- Define the server, database, and Agent security measures and ensure that Agents are run in the proper context.
- Code Agents run by the Form’s WebQuerySave event to translate and validate user entries and provide useful feedback to users and redirect them to natural landing points in the application.
- Code Agents run by the `?OpenAgent` URL command to perform powerful actions and produce reports from Domino data, and learn different ways that a browser can run the `?OpenAgent` command.
- Debug Agent code and add run time error handling routines as well as how to analyze Agent-specific browser errors.
- Define the key components and protocols of Web Services technology.
• Use Web Services Explorer to perform basic testing of a Web Service.

• Describe the differences between the various Web Services programming models and SOAP message formats as they apply to Web Services design element properties.

• Code Web Services using LotusScript that accept simple and complex data types as parameters.

• Code a simple Web Services client that runs from a browser and uses a SOAP toolkit.

• Import a WSDL to create a LotusScript library that can be used by Notes or Domino to consume a Web Service.

• Debug Web Services provider and client code and handle runtime code errors as well as SOAP faults.

• Employ the Remote Debugging, Agent Logging, Profiling, and DDM Probe features to fully analyze Agent and Web Service operation and performance.

---

**Audience**

This course is designed for experienced programmers well versed in using Domino Designer 8. It assumes that you have:

• thorough knowledge of the Domino Designer 8 development environment, including Form and View design, as well as how to use properties boxes, and how to set the ACL

• mastered the Topics covered in these courses:
  
  • *Domino Designer 8: Basic Notes Applications*
  
  • *Domino Designer 8: Basic Browser Applications*

• knowledge of Web technologies, including servers, browsers, HTML, Cascading Style Sheets, and some basic JavaScript.

While mastery is not expected, you should have a working understanding of LotusScript and the various Notes product objects.
This course is part of a series of Domino Designer 8 training courses. Follow these paths to master all aspects of developing applications using Domino Designer:

**Domino Designer 8: Basic Notes Applications** provides the base knowledge for this additional training:

**Client Track**

- Configure Domino to use DB2 as a database engine and build applications that access relational data, *DB2 for Domino 8 Administrators and Developers*.

- Specialize in programming Notes applications to respond to built-in user interface features, *Domino Designer 8: Special Notes Features*.

- Convert an application written for Notes clients to be used in browsers, *Domino Designer 8: Basic Browser Applications*.

- Provide data to cutting-edge Web applications, *Domino Designer 8: Web Agents and Web Services*.

**Language Track.** These languages apply to both Notes and browser clients:

- Learn sophisticated application techniques that fully exploit the formula language in *Domino Designer 8: Formula Language*.

- Develop sophisticated applications and Agents with LotusScript in *Domino Designer 8: LotusScript*.

- Add powerful client-side scripting to browser applications in *Domino Designer 8: JavaScript*.
Course design

This is a lecture/lab course in which you will closely examine demonstration code and either modify or create additional Agent and Web Services code. You will immediately apply the concepts and techniques as you work in your own project database.

Data files

The project database, two HTML files, and two toolkits accompany this course and can be installed by running the INSTALL.EXE that comes with this course. (You can also download the file from http://www.waresource.com/.) The files contain working examples of the techniques described in the course. The files include:

- **DDWAWS8 Demo (DDWAWS8Demo.NTF)**, the starting template for the project application that you will continue building in this course.
- **PocketSOAPDemo.HTML**, the source file for the Web Services client used for demonstrations and further coding exercises.
- **SOAPFaultDemo.HTML**, which demonstrates how to handle SOAP faults.
- **PocketSOAP.1.5.4.exe**, included for convenience. The SOAP toolkit you will use in course to demonstrate and further code a Web Services client.
- **proxyTrace.exe**, included for convenience. Captures SOAP traffic from the Web Services client for closer analysis of request and response messages.

Install the files in the DDWAWS8 directory under the default Notes data directory, for example, c:\LOTUS\NOTES\DATA\DDBBA8\.

During the course you will create your project database on the Domino Server in the classroom using the starting template.

Be sure to have access to Domino Designer 8 Help, which should be full text indexed for quick searching.

You will also need 1.5 level or higher Sun Java 2 Runtime Environment (JRE) or Java development kit (JDK) and IBM’s Eclipse 3.3+ and the Web Tools Platform 2.0.1+ installed on your computer. You will do this in an exercise in the course or to save time, the Instructor may have already installed the software for you.

Please consult the Set Up document for this course to make sure the correct environment is in place before starting the course.
Conventions

This course follows these font conventions:

- *Italic* - database, View, Page, Form, document, macro, and Field names, as well as object event types
- **Bold** - Notes menu options, command button names (whether Notes or developer defined), and accelerator keys
- **Courier** - user input, sample values, code examples (which are often marked with “Fragment” numbers in the text)
- **Helvetica** - HTML and JavaScript code examples
- ® - shows when lines of code wrap in this text but should be one continuous line in the Programmer’s Pane.
Topic 1: Introduction

Key points

A Domino Agent triggered by a browser and a Domino hosted Web Service have many things in common, primarily that they are both run by the HTTP task in response to a client request.

The request may ask the server to perform some sort of action, such as to update a document, or return something appropriate to the client, perhaps the result of a query, the output from a function, or a page of dynamically generated HTML or XML.

Agents

There are three types of Agents covered in this course, all related to applications run by browsers over the Web. All three use the same coding techniques, can be written in LotusScript or Java, and use the same security model. They differ only in what they hope to accomplish and how they are triggered:

- **WebQueryOpen event Agent.** Run before the Form is converted to HTML and sent to the browser.

  This Agent is used to set default Field values, perhaps to build a default Keywords list from a lookup, increment a counter to determine how many “hits” a page has received, or create a record of the user’s visit to the page.

  No output (in the form of Print statements) is allowed, as the Form always opens at the conclusion of the Agent. The Form itself may have JavaScript in its onLoad event that redirects the user to another address, perhaps set by the Agent in the WebQueryOpen event.

- **WebQuerySave event Agent.** Run after the browser user submits a Form but before the document is saved to disk.

  This Agent performs Field validation and is capable of performing complex tasks involving the current document or other documents in the same or other databases.

  Any output using a Print statement is returned to the browser as text. The text may be unformatted plain text or it can contain the necessary tagging to return HTML (including embedded JavaScript) or XML (Extensible Markup Language) to the browser, or redirect the user to another URL.
• **“Web Agent”** (for lack of a better term to distinguish from the other two types of Agents triggered by a browser). Invoked by some browser event such as a Form event or button click that results in a URL that includes the ?OpenAgent command.

Web Agents are used to perform tasks, such as perform lookups or validate the Form using backend methods and data that are otherwise not available to JavaScript validation techniques, delete the document, operate on multiple documents, redirect the browser to another location, etc.

Web Agents can also produce any type of textual output back to the browser using Print statements.

---

**Web Services**

The HTTP task also responds to client requests for Web Services that were created by Domino Designer 8 and stored in an NSF database. Web Services design elements resemble Web Agents in how they are designed and secured, and like Web Agents, can be written in either LotusScript or Java. In essence, both Web Agents (the non-Form event type) and Web Services are a type of remote procedure call to the server, able to both request data from and run procedures on the server, and return a text-based response to the client.

What distinguishes Web Services from Web Agents (the non-Form event type) is that the request for a Web Service is in the form of a standards-based XML document wrapped in SOAP tags. The response, if there is one designed into the Web Service, is also in the form of XML wrapped in SOAP, which the client must unwrap before processing the XML to extract any data or return codes.

---

**What follows…**

In the Topics that follow, you will discover what is common to all three Agents run from browsers while learning about the Form WebQueryOpen event.

You will then learn what is unique about Agents called from the WebQuerySave Form event and those called from the ?OpenAgent URL command.

Attention will then turn to the Web Services design element, starting with the basic concepts of Web Services. From there you will see how to create a Web Service in Domino Designer, as well as how to test and use one from several types of clients including Notes and Domino.
Exercise: Create demonstration database

All of the examples shown in this course can be found in the *DDWAWS8 Demo* database. This is also the database you will modify as you work on the various exercises in the course.

Follow these steps to create your own working copy of the *DDWAWS8 Demo* database on the server.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Instructor has registered you as a Domino user and has given you the right to create new databases on the server. The Instructor will provide your Notes user name, Notes User ID password, and Internet Password (will be the same as your Notes password).</td>
</tr>
<tr>
<td>2</td>
<td>Working in Notes or Domino Designer, create your own working copy of the <em>DDWAWS8 Demo</em> database (from the <em>DDWAWS8Demo.ntf</em> template) on the classroom server.</td>
</tr>
</tbody>
</table>

**Note:** The *DDWAWS8Demo.ntf* template can be found locally after running the class data disk (*INSTALL.EXE*) but the Instructor has also placed the template on the classroom Domino Server.

Choose **File – Application – New.**

It is critical that you:

- Create the new database on the Domino Server.
- Enter a new title and file name that includes your name or initials to uniquely distinguish it from other databases on the server.
- Select the *DDWAWS8 Demo* template on the Domino Server as the source of the design.
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 3    | For example:  

![New Application dialog box](image)  

Click **OK** to create the copy on the server. |

| 4    | Open the ACL and make these changes:  

- Add a colleague and give him/her Author access, can create/delete documents. This entry represents a typical user entry for this application that you will use to test your work.  

- Verify that the classroom Domino Server is set as Manager.  

- Verify that the –Default– and Anonymous entries are set to **No Access**. Select these two Attributes: **Read public documents** and **Write public documents**. |
Now that your working copy of the *DDWAWS8 Demo* database is on the server, test access with a browser using a URL such as this one:

```
http://<serverhostname>/JoeDDWAW.nsf?open&login
```

Enter your name and Internet Password. The application should open to the default browser interface listing the two Views:

![Views](image)

We have kept the application design simple so you can focus on the code and not worry about the user interface.
Topic 2: WebQueryOpen Event Agents

Key points

This Topic describes the purpose of the Form WebQueryOpen event and how the Agent called by the event operates.

This Topic also covers the design of Agents run by the WebQuerySave event and the OpenAgent URL command.

Form computation sequence

To understand the purpose of the WebQueryOpen event you must first understand its place in the Form computation sequence for Domino applications running in a browser:

The FormEval Form demonstrates two characteristics of how the WebQueryOpen event Agent operates in the context of how the Form is computed:

- The Agent runs after the rendering engine has computed all the Field formulas, so if you reset any Field values from the Agent, they won’t be recomputed by the Field formulas until the document is refreshed or incrementally saved.

- With respect to computed Subforms dependent on a Field value to determine which one to load, the WebQueryOpen event Agent can set the Field but the Subform won’t be loaded until after the document is incrementally saved and the HTML form is refreshed.

- The Agent runs before the rendering engine converts the Form (and document data) to HTML, but after all the Field formulas are evaluated. As such, the Agent run by the WebQueryOpen event has full access to data in the document.
If desired, the WebQueryOpen event Agent allows the Form to be entirely dynamic in that it can write out any valid HTML and/or JavaScript combination to pre-defined Fields.

A WebQueryOpen event Agent is entirely optional. In fact, the Field default, translation, and validation can be done more efficiently using the Field formulas as opposed to the server finding, loading, and running the external Agent.

---

**Step 1: Create the Agent and set its properties**

All three types of Agents run from browsers are configured using these Runtime properties:

The Agent must have these properties set:

- **Be Shared.**

- Trigger set to run **On event** from the **Agent list selection** (preferred so it is not seen by Notes users in the **Actions** menu).

- Runtime Target set to:

  - **None** if the Agent is triggered from the WebQueryOpen/WebQuerySave events or is an Agent run by the `?OpenAgent` command.

  - **All documents in database** if the Agent is run apart from any Form using the `?OpenAgent` command.

In either case, the code (not Agent properties) is responsible for further refining which documents to operate on.
You will look at the **Security** tab below.

**Example:** Open the *MainWQOn* Agent to view its Agent properties.

**Note:** Agents triggered by a browser run only from an `.NSF` file (they won’t run from an `.NTF`).

---

**Step 2: Code the Agent**

Agents run from a browser can be coded in LotusScript or Java. LotusScript will be used in this course for simplicity’s sake.

You can perform any number of operations in the code and access:

- Domino objects (backend classes only; no UI or NotesTimer classes are allowed in any of the three types of Agents run by the HTTP server task or in Web Services, for that matter)

- data in the existing document as well as in any document in the current database or other databases on the server or on other servers

- external objects (e.g., COM from LotusScript or non-Domino classes from Java)

- external data sources using IBM Lotus Enterprise Integrator for Domino (LEI), Domino Enterprise Connection Services (DECS), Lotus Connector LotusScript Extensions (LC LSX), or a third party bridge.

**No global variables available**

There are no global variables that persist from the WebQueryOpen event Agent to HTML Form events or to the Agent that is called by the WebQuerySave event Agent. You can, however, use these two methods to emulate a global variable:

- **Use the current or some other document Fields (often hidden from the user) to pass values.** If the Fields are not visible in the Form, it is essential that you set the **Generate HTML for all fields** Form property so that the values will be available to the WebQuerySave event Agent after the HTML form is submitted.

- **Use a Profile Document keyed to the user name** (Agent must be set to “Run as web user” on the Agent properties **Security** tab, described below, to distinguish between users). In some cases you can carefully use a non-keyed Profile Document, such as to increment a sequential number shared across all users (as long as you distinguish new documents from existing ones using the branching techniques based on document state, also described below).
Reference the current document

The most-used code technique in Agents run by the WebQueryOpen event is the ability to access the current document object using the DocumentContext property of the NotesSession class. Once the current document is instantiated as an object, you then have access to the document Fields, for example via the extended class syntax:

```vba
Sub Initialize
    Dim s As New NotesSession
    Set doc = s.DocumentContext
    doc.Status = "New Document…"
End Sub
```

**Example:** Open the `MainWQOn` Agent to view the code in the Initialize event, which sets “doc” to the current document object using the DocumentContext property of the NotesSession object.

Set Computed or Computed when Composed Fields

By default, if you set Computed or Computed when Composed Fields from the Agent run by the WebQueryOpen event, the values are not saved to the document unless you have the **Generate HTML for all fields** property set on the Form properties **Default** tab.

**Example:** Open the `Main` Form to view its Form properties. Click the **Defaults** tab to see that the **Generate HTML for all fields** property is set. Notice that the **Status** Field is Computed when composed.

Open the `MainWQOn` Agent to view the code in the Initialize event. This line sets the **Status** Field:

```vba
doc.Status = "New document…"
```

Working in your browser, create and save a new `Main` document. Switch to Notes to see that the value of the **Status** Field is “New document…”.

**Caution:** Be aware that the **Generate HTML for all fields** property presents possible security issues with users being able to see values in hidden Computed Fields by looking at the HTML source.
DelayUpdates property for Agents

When performing many saves and removes on NotesDocument objects in an Agent, you can decrease the amount of time the Agent runs by setting the NotesDatabase object DelayUpdates property, for example:

```vbs
Dim s As New NotesSession
Dim db As NotesDatabase
Set db = s.CurrentDatabase
db.DelayUpdates = True
```

With this property enabled, updates to the documents are cached and the Agent continues. The changes are committed when the server has free resources.

The property is not saved in the database, so you must set it in each script.

**Warning:** Delaying updates means that you risk losing document changes should the server crash before changes are written to disk.

No direct output is allowed

You cannot use Print to output to the browser from an Agent run by the WebQueryOpen event because when the Agent finishes, the HTML Form is opened for the user. Any Print statements are ignored.

Of course because you can write HTML to the Form via a Field, you can use JavaScript to produce any output or to redirect the browser, for example as seen in the MainWQOn Agent:

```vbs
Dim s As New NotesSession
Set doc = s.DocumentContext
doc.HTML="<script>location.href="http://www.yahoo.com"</script>"
```

When the browser opens the Form and processes the HTML Field (Computed for Display), it finds executable JavaScript that redirects the user to another URL.
Topic 2: WebQueryOpen Event Agents

Step 3: Call the Agent from the Form’s WebQueryOpen event

Although the Agent is written in LotusScript or Java, it is called by @Command([ToolsRunMacro];"AgentName") in the Form’s WebQueryOpen event, for example:

![WebQueryOpen event diagram]

Why do the WebQueryOpen and WebQuerySave events call an Agent? Because the same Agent can be called by multiple Forms that all use similar routines, thereby reducing your code maintenance efforts.

Distinguish between document states

The Agent called by the Form WebQueryOpen event runs whether the document is being created for the first time, opened for editing, or opened for reading. This presents a potential of overwriting Field values entered by users if the same code runs for all three document states. There are two ways to overcome this potential problem:

- Use @If in the WebQueryOpen event to run a different Agent for each document state. In this example from the Main Form, one Agent runs if the document is new, another Agent runs when the document is being edited, and a third runs when the document is opened for reading:

![Main Form WebQueryOpen event diagram]

You can use any non-UI @Functions to determine which Agent to run using the ToolsRunMacro command.
• Branch within the Agent code itself depending on the document state, in this example from the ReviewWQO Agent:

```livescript
If doc.IsNewNote Then
    doc.Status = "New document..."
Else
    doc.Status = "Opened for editing or reading..."
End If
```

The disadvantage of this technique is that because there is no UIDocument object open, LotusScript cannot distinguish between a document opened for editing or reading.

**Call multiple Agents**

You can also call multiple Agents, which run in sequence. For example, look at the WebQueryOpen event in the FormEval Form. It runs two Agents in sequence:

For demonstration purposes, the first Agent, FEWQO1 sets the first Field on the Form; FEWQO2 sets the second one.

Though tempting to compartmentalize operations into multiple Agents, calling more than one Agent adds to the overhead of the Form, resulting in a slower load time for the user. So avoid doing this if at all possible by putting all the code into one Agent.

---

**Agent authority**

Agent security builds on the browser-based application security model:
Agents cannot circumvent the Domino security mechanisms. When run in context of a Form event, an Agent is also subject to the security mechanisms of the Form.

Understanding Agent security is not easy, as it involves how users log in to the server, how the Agent is signed, and whether the user’s or the Agent signer’s authority is used to control the Agent’s operations.

**Authenticated user names**

Users run Agents from a browser with either Anonymous access to the server or as an authenticated user. Domino uses identity-based security, meaning that the name of the user derived from the authentication process is the basis for all other security mechanisms, such as server access or database ACLs.

At the end of the authentication process, the user name known and trusted by Domino is an *authenticated* user name. But the name entered by the user during the login process may not necessarily be the final authenticated name!

When users enter their name and password, they actually have a degree of latitude in how they enter their names. The Domino administrator defines the allowable login names in the *Server* document in the *Domino Directory* with the *More name variations with lower security* option. To limit users to just entries in the *UserName* and *AltUserName* Fields in the user’s *Person* document, the administrator should select *Fewer name variations with higher security* instead.

Take, for example, the *Person* document for a user named Joe Smith:

![Person Document](image)

Two things to note here:

- The alias name “Joey” was manually added after registration by the administrator for convenient Notes mail addressing to this user.
- Users who are self-registered using a registration application will NOT have a distinguished name component (e.g., ”/TeamApps”).
This table describes the two name variation settings and how they impact authentication:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>More name variations with lower security</td>
<td>Default option that uses any value in the hidden $Users View in the Domino Directory. Poses a security risk because more name combinations can be authenticated. Users can enter any of these name components:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Last name:</strong> Smith</td>
</tr>
<tr>
<td></td>
<td>• <strong>First name:</strong> Joe</td>
</tr>
<tr>
<td></td>
<td>• <strong>Common name:</strong> Joe Smith</td>
</tr>
<tr>
<td></td>
<td>• <strong>Full hierarchical name (canonical):</strong> CN=Joe Smith/O=TeamApps</td>
</tr>
<tr>
<td></td>
<td>• <strong>Full hierarchical name (abbreviated):</strong> Joe Smith/TeamApps</td>
</tr>
<tr>
<td></td>
<td>• <strong>Short name:</strong> jsmith</td>
</tr>
<tr>
<td></td>
<td>• <strong>Alias name</strong> (a name listed in the User Name Field of the Person document, excluding the first name listed in the field): Joey</td>
</tr>
<tr>
<td></td>
<td>• <strong>Soundex number:</strong> s530 (from the hidden $Users View in the Domino Directory)</td>
</tr>
<tr>
<td>Fewer name variations with higher security</td>
<td>The recommended setting (uses the hidden $LDAPCN View) because it reduces the possible number of names that can be used to authenticate. Users can enter only of these name components:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Full hierarchical name:</strong> Joe Smith/TeamApps</td>
</tr>
<tr>
<td></td>
<td>• <strong>Common name:</strong> Joe Smith</td>
</tr>
<tr>
<td></td>
<td>• <strong>Common name with CN= prefix:</strong> CN=Joe Smith</td>
</tr>
<tr>
<td></td>
<td>• <strong>Alias name</strong> (a name listed in the User Name Field of the Person document, excluding the first name listed in the field): Joey</td>
</tr>
</tbody>
</table>
Note: Regardless of the “name variations” setting, the login name is case insensitive. (Passwords, of course, must be exact.)

The point is that once authenticated by whatever mechanism, and regardless of which name variation option is enforced, Domino returns ONLY the first value in the User Name Field in the Person document. In this example, Joe Smith/TeamApps is the first value, and is the precise name that must be used on all security mechanisms such as Group document member lists, Server Access Lists, database ACLs, and Authors/Readers Fields.

The authenticated user name is also the one used when the HTTP server task determines if the user is allowed to run an Agent.

Tip: To ensure that you specify the first value in the User Name Field when applying security mechanisms, always use the address picker when selecting users to include in groups or ACLs. Do not manually type any names.

Note: Name and password authentication is only one possible mechanism. The Domino HTTP server task also supports authentication via LDAP, single sign on, digital certificates, or DSAPI. Depending on the transport protocol, additional credentials may be required (e.g., over HTTPS users may need a client-side digital certificate to authenticate with the server).
Anonymous users

In some cases, the server will allow anonymous access over the HTTP protocol, so authentication isn’t necessary, and as a result all users will be named “Anonymous”. This is set in one of two places:

- A Server document that does not “Load Internet configurations from Server\Internet Sites documents” (set on the Basics tab) on the Ports\Internet Ports tab:

![Server document settings](image)

This is a setting used on public read-only servers or on a server set aside for Anonymous users to create a user name and password.

When a server is configured to allow Anonymous access, it is even more critical for database ACLs to be carefully configured to protect sensitive data.

- In a Web Site document in the Web\Internet Sites document if Server\Internet Sites documents are used, for example:

![Web Site document settings](image)
Agent signature

When you save an Agent design (or click the Sign button in the Agents list), your Notes ID is used to sign it.

The Agent won’t run if your Notes ID file was not created from the same Organizational Certifier as the Server ID that will run the Agent.

If you are developing the Agent for another Organization, the Agent must be re-signed before it will be allowed to run on the server, and the server must provide that signatory with the necessary execution rights. Agents are re-signed when saved or signed using the Sign tool on the Files function tab in Domino Administrator.

**Warning:** Although possible, it is highly recommended that you do NOT sign any design element with the Server ID. This is a huge security risk, as the server has total access to all databases.

Authorization to use server

The Server document controls general access to the server over HTTP. If the Ports\Internet Ports\Web tab field Enforce server access settings is set to Yes, the effective user must be given permission on the Security tab to access the server and, as appropriate, to run various types of code in the Programmability Restrictions section.

Step 4: Set the Agent properties Security tab options

The Agent properties Security tab specifies under whose authority the Agent will run.
You have three options as to whose credentials are used to run the Agent:

- The person who signed the Agent, typically the developer or administrator (Run as web user and Run on behalf of properties are not set).

- **Run as** [an authenticated] **web user**, as shown above.
  
  Under most circumstances, you want to set this property so that the Agent is run using the credentials of the authenticated Web user. This is more secure in that it restricts the user to performing only those things that the database ACL would let the user do apart from the Agent.

- **Run on behalf of** another user.
  
  This setting is almost always used exclusively by the Out of Office Agent in Mail. It allows the Agent to run with the authority of the Agent signer, but documents created by the Agent and any messages sent by it are stamped with the “on behalf of” name as the author and sender. But you may also find use for this setting for Agents that provide just read-only operations or if there is no authentication mechanism possible by the client.

The choice you make determines the “effective user name,” which in turn determines the authority the Agent will have when it performs operations in the database, such as to create, modify, or delete documents.

**Restricted operations**

The **Set runtime security level** setting on the Web Service Security tab further restricts the permissions to run unrestricted operations, as set in the Server document on the Security tab.

Restricted operations are those calls in Java and LotusScript that impact the underlying operating system on the server, such as to use a disk-based log file, set environment variables, manipulate an Embedded Object, use file I/O or network I/O, set system date/time, call procedures in an external .DLL, or execute another application.

The Agent properties Security tab has three options with respect to limiting the Agent’s ability to use restricted operations:

- **Do not allow restricted operations**, the most secure choice.

- **Allow restricted operations**, which is less secure because it gives the Agent more power. The caveat here is that the effective user name must also be given unrestricted rights in the Server document or the restricted operations in the Agent will not run.
• **Allow restricted operations with full administration rights.** Not only does the Agent have the power to run restricted operations, but it also has full administration rights on the server, basically giving it absolute power over everything on the server.

Most likely the group of developers has unrestricted rights in the *Server* document (*Programmability Restrictions* section), but for most Agents, you can/should limit those rights to disallow restricted operations.

The net result is that even if a user may have been given the right to run unrestricted methods and operations in the *Server* document, the Agent developer/administrator can put limits on what actions the Agent can actually perform.

Conversely, you cannot elevate a restricted user (as specified of the *Server* document) to use restricted methods by telling the Agent to allow restricted operations.

If in doubt, always leave the default setting, **Do not allow restricted operations**, to prevent the Agent (and the effective user who otherwise has unrestricted access to perform restricted operations on the server) from having too much power.

The **Default access to this web service** and **Allow Public Access users to use this web service** settings on the Web Service Security tab refine the permissions set in the database ACL.

---

**Who can run the Agent**

The setting **Default access for viewing and running this agent** allows you to restrict who can run the Agent, whether in the context of a Form event or **OpenAgent**.

Whether or not a user can use the Form also determines indirectly if the user can run the Agents called by the WebQueryOpen and WebQuerySave events. The list of users authorized to run the Agent is checked when the Agent actually runs, regardless of the context from which it is called.

If not set to **Allow readers and above**, who should be listed? In addition to the list of authorized users, developers, and administrators who can run the Agent, be sure to include the name of your Domain’s Server group, typically **LocalDomainServers**. If you neglect to do this, the Agent design element will not replicate to other servers that may also be hosting the application.

Allowing Public Access users to run the Agent requires that their ACL entry be set to have Public Access.
Database ACL

Regardless of whose credentials are used when the Agent runs, the database ACL ultimately controls the level of access to the data that the Agent’s effective user has. At the very least, the database ACL must give the effective user Depositor access with the right to Read public documents. If the Agent is programmed to create, edit, or delete documents, higher access is required.

Let’s give three examples:

- **Joe Smith/TeamApps** logs into the server using his name and password. An Agent is set to **Run as web user**. When the Agent attempts to change any data in the database, the database ACL is checked to see if the name “Joe Smith/TeamApps” has the authority to make the desired changes.

- An Internet-facing Domino Server allows anonymous connections. A user opens an application on the server where Anonymous is given Reader access in the database ACL. The Agent is also set to **Run as web user**. But because the Anonymous user only has Reader access, any attempts by the Agent (or URL hacking) to change document data is prohibited.

- Two users open an application on the same Internet-facing Domino Server that allows anonymous connections. One user comes in as Anonymous, but the other one logs in with a name and password. An Agent is NOT set to **Run as web user**, which means it runs with the authority of the Agent signer. The Anonymous user only has Reader access in the ACL, so any attempts by the Agent to change document data is prohibited. The authenticated user has Author access in the database ACL (and can create documents), so the same Agent is able to create documents (assuming that the Form access list allows it).

**Caution:** The only time you would want to allow Anonymous users to have Author access in a database ACL and be allowed to create documents is if your application model doesn’t allow them to edit documents. This is because Domino doesn’t distinguish between one Anonymous user and another one. So if everyone is “Anonymous” then everyone can edit documents created by every other Anonymous user!

**Note:** While working in the ACL, be sure to verify that the Maximum Internet name and password setting on the Advanced tab is set just high enough to let users perform the required actions in the database. This refines/limits the access given on the Basics tab of the ACL, but cannot elevate any entry’s access level.
**Exercise: Set default email address in Review form**

As it currently stands, the user must manually type in an email address into the *Review* Form. To make it easier for users, the goal of this exercise is to have the Agent run by the WebQueryOpen event find the user name in the *Domino Directory* and return the InternetMail address.

Follow these steps to add a default email address for authenticated users when they open a new *Review* Form:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the <em>Review</em> Form in Domino Designer.</td>
</tr>
<tr>
<td>2</td>
<td>Expand the Form events to find the WebQueryOpen event.</td>
</tr>
<tr>
<td>3</td>
<td>Notice that the event calls the <em>ReviewWQO</em> Agent.</td>
</tr>
<tr>
<td>4</td>
<td>Open the <em>ReviewWQO</em> Agent design.</td>
</tr>
<tr>
<td>5</td>
<td>You will find that the SetEmailDefault function has already been added for you. Take a look at the code to see how it finds the current user’s InternetMail address from the <em>Domino Directory</em>. Also notice in the Agent properties Security tab that the Agent runs as a web user. Very important, otherwise the Agent signer’s email address would always be returned.</td>
</tr>
</tbody>
</table>
| 6    | Add this line to the bottom of the Initialize event (right above the End Sub) to call the Sub:  
   ```vbscript  
   Call SetEmailDefault(s, doc)  
   ``` |
<p>| 7    | Save the Agent and test the Form from your browser. Your InternetMail address should appear as the default for the email address Field. |
| 8    | The problem with the <em>ReviewWQO</em> Agent with respect to the WebQueryOpen event of the <em>Review</em> Form is that if another user tries to edit a document created with the Form, that user’s email address will overwrite the first user’s email address. |
| 9    | To overcome this problem, you will need to create a copy the <em>ReviewWQO</em> Agent and rename the copy to <em>ReviewWQOe</em>. The “e” stands for “editing.” |</p>
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 10   | Change the *ReviewWQOe* Agent, removing this line of code:  
                  *Call SetEmailDefault(s, doc)*  
                  Also delete the SetEmailDefaults Sub. |
| 11   | Save and close the *ReviewWQOe* Agent. |
| 12   | Open the *Review* Form design. |
| 13   | Modify the WebQueryOpen event formula so that it calls the *ReviewWQO* Agent for new documents, and the *ReviewWQOe* Agent if the document is being edited (use @IsDocBeingEdited in an @If statement). |
| 14   | Save and close the Form design. |
| 15   | Open the *AllDocs* view in a browser.  
                  If working alone, edit an existing *Review* document that was added as part of the template. Notice that the agent does not overwrite the email address of the original author with your email address.  
                  If working with a colleague, as that person to create a new *Review* document in your exercise database. (You may have to add that person to the ACL as an Author, Can create documents.) Then edit the review to prove that your email address does not overwrite your colleague’s email address. |

**Agents and document access Fields**

Agents run from a browser also are subject to any document-level restrictions. So even if a user has the necessary authority to read or edit a document according to all the other security measures, access to edit or delete specific documents may be restricted.

The **Security** tab on Form properties lets you define who can read documents created by the Form, as well as users who are allowed to create documents using the Form.

In addition, you can set Readers and Authors Fields on the Form to restrict access to the document data even further. This is usually done in the Field formulas, but can also be set programmatically via LotusScript (for example in a workflow or routing sequence).
Example

Open the *Main* Form to find the *DocAuthors* Field. This Computed when composed, multi-value, Authors Field is actually set programmatically by the *MainWQOn* Agent. Open the Initialize event of the *MainWQOn* Agent to find this code:

```vbscript
Dim newValues(1 to 3) as String
newValues(1) = s.EffectiveUserName
newValues(2) = "LocalDomainServers"
newValues(3) = "Administrators"
Set authorsItem=doc.ReplaceItemValue("DocAuthors", newValues)
authorsItem.IsAuthors=True
authorsItem.IsSummary=True
```

The *DocAuthors* Field is set by the Agent when the document is created to the value of the current effective user (the name of the user if the *Run as web user* property is checked in Agent properties or the name of the person who last saved the Agent if not), as well as the Administrators and LocalDomainServers groups. Including LocalDomainServers ensures that the document will replicate to other Servers.

Prevent Agents from being run out of context

First things first. If any particular Agent is never to be run from the Web, you should hide the Agent design element from Web browsers (set on Agent Design Document properties).

Then it is highly recommended that you code all Agents so they cannot be run using the `?OpenAgent` Domino URL syntax outside the context of a document (either as a link on the page or called by the WebQueryOpen or WebQuerySave events).
To prevent a user from running an agent by hacking a URL, add this code to the beginning of all Agents:

```
If Not(Instr(1, Ucase(doc.HTTP_Referer(0)), Ucase(doc.Server_Name(0))) > 0) And 
Not(Instr(1, Ucase(doc.HTTP_Referer(0)), Ucase(doc.HTTP_HOST(0))) > 0) Then

   Print {<HTML><HEAD><TITLE>Error</TITLE></HEAD><BODY>}

   Print {<H1>Error</H1>Unauthorized Exception: You cannot run the Agent from a URL.<P><HR>}

   Print {</BODY></HTML>}

Exit Sub

End If
```

(Actual code goes here...)


Though written in the R5 timeframe and some of the problems discussed were fixed in later versions (e.g., the elimination of $DefaultNav), there are still some valuable lessons to be learned from the paper.

**Exercise: Prevent Agents from being run out of context**

Follow these steps to see how the code in the *ReviewWQO* Agent prevents users from running the Agent from a URL:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1    | Open the *Review* Form in your own working copy of the project database using a URL such as:  
| 2    | The *ReviewWQO* Agent run by the WebQueryOpen event sets the *Status* Field to “New document.” |
| 3    | Now try to run the same Agent directly using a URL such as:  
<p>| 4    | The error page opens instead of the <em>Review</em> Form, preventing you from running the <em>ReviewWQO</em> Agent out of context. |</p>
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Working in Domino Designer, open the <code>ReviewWQO</code> Agent.</td>
</tr>
<tr>
<td>6</td>
<td>Locate this line of code:</td>
</tr>
<tr>
<td></td>
<td><code>Call CheckContext(doc)</code></td>
</tr>
<tr>
<td>7</td>
<td>Trace the code source by opening the <code>CheckContext</code> function.</td>
</tr>
<tr>
<td>8</td>
<td>The function is passed a handle to the current document context and checks to see if it is being run from the current server and database. If not, the user is kicked to an error page.</td>
</tr>
</tbody>
</table>

**Note:** One thing that may strike you odd here. It was said earlier that you cannot redirect users with a WebQueryOpen Agent. But it looks here as if the `ReviewWQO` Agent is in fact redirecting a user! No, this is not a contradiction. Remember that in this exercise you attempted to run the Agent from the `?OpenAgent` command, which means the rules for being a WebQueryOpen Agent don’t apply.