TOC

About This Guide

This is an archived copy of the v23 documentation provided for Logi Info v23.3 and its service packs.

Notice: Archived Documentation

This documentation is provided as a courtesy reference for a version of our software that is no longer under active development or support. The information contained herein is offered without warranties of any kind, either expressed or implied, including but not limited to warranties of accuracy, completeness, or fitness for a particular purpose.

While this archived material may assist with understanding historical functionality, please be aware that the software described is no longer maintained at this version level and may contain outdated or inaccurate information. Images may not reflect currently supported modules, support sites, or third party products. This software may not be compatible with current versions of previously compatible third party products.

To access and upgrade to current software solutions and receive ongoing support, please contact our customer support team. They can assist you in migrating to the latest appropriate software version that meets your needs. Our support team is available to help ensure a smooth transition to actively maintained alternatives that provide the functionality and reliability you require.

Secure Logi Info Applications

Logi Info includes features that allow you to build applications that securely control access to reports and data, and prevent malicious users from adversely affecting the system.

The following topics provide an overview for application developers who want to build secure Logi applications:

- Security, Authentication, and Authorization
- Session Management
- Hardening a Logi Application
- Hardening the Application Environment
- Obfuscating Definitions

Security, Authentication, and Authorization

Logi Info provides a flexible mechanism, "Logi Security", for implementing and enforcing security in Logi applications. It's capable of authenticating users, enforcing roles, enabling different security privileges for different users, and controlling web page component visibility. It can also control data access, down to the row-, column-, and cell levels. Logi Security is described in detail in "Logi Security" on page 25.

Logi Info applications track every user's authentication status for each web request made. The best practice is to employ the **Security** element in the _Settings definition; it provides proper authentication and support for highly granular authorization to access reports, functionality, and data. The Security element is described in "Working with Logi Security" on page 37.

Logi **SecureKey** authentication technology provides integration for applications requiring secure access to a Logi application or embedded Logi components. It enables a "single sign-on" environment while enhancing security: user authorization is established and requests are made to Logi applications or embedded components using a special security key. SecureKey is discussed in "Logi SecureKey Authentication" on page 77.

Logi Security works with a variety of security scenarios. Specific examples of application and web server configurations for seven different scenarios is available in "Logi Security Scenarios " on page 59.

Session Management

Developers may want to exert control over the time span of a user's application session. This can be done by configuring an appropriate session timeout period for applications and, in the case of embedded Logi applications or components, may need to be synchronized with a parent application. The application's session timeout should match the timeout configuration of the application server.

"Session Variables" on page 133provides insight into how these variables are used and how Java and .NET applications can share them. Logi Info also provides a **Session Timeout** element, described in "Managing Session Timeout " on page 145, to assist with management of the user session.

Hardening a Logi Application

Here's a list of Best Practices we recommend to you when building a secure Logi application:

- When retrieving SQL data, use the **Handle Quotes in Tokens** attribute available in DataLayer.SQL and DataLayer.ActiveSQL. These ensure that strings with quotes are handled appropriately. This is discussed in *DataLayer.SQL*.
- Use the **SQL Parameter** element to pass tokenized parameters, rather than embedding them directly in the SQL statement. For more information, see *DataLayer.SQL*.
- Use stored procedures, whenever available, to enable applications to validate user inputs in stored procedure execution. This is discussed in *DataLayer.SP*.
- Validate user input to ensure it is in the proper format. Validation elements are available for this purpose and are discussed in *Work with User Input Elements*. Any custom JavaScript used in validations should be reviewed for potential vulnerabilities.

Apply Request Filtering in the application configuration to secure against potentially malicious text inputs or attempts to directly browse source code. For example, this code in your Logi .NET application's web.config file blocks direct browsing of anything in the _Definitions folder, in IIS 10:

```
<system.webServer>
<security>
<requestFiltering>
<filteringRules>
<filteringRule name="BlockDefinitionAccess" scanUrl="true" scanQueryString="false">
<denyStrings>
```

```
<add string="_Definitions" />
</denyStrings>
</filteringRule>
</filteringRules>
</requestFiltering>
</security>
</system.webServer>
```

Java-based servers also have Request Filtering features that can be used in a similar manner. Please review the documentation available for your application web server.

- Use JavaScript for type-checking file uploads performed in the Logi application.
- Use a proxy module to either whitelist or blacklist information passed via Logi request parameters.
- Use the HTTPS protocol instead of HTTP for all application web traffic.

Hardening the Application Environment

Many web application servers are attacked every day. The best defense against such attacks is to ensure that server hardening is a well-established practice within your organization. All devices and software that are involved in deploying and supporting a web application, such as servers, firewalls, databases and others, must be securely configured. Some common server hardening tips & tricks include:

- Avoid using insecure protocols that send your information or passwords in plain text.
- Minimize unnecessary software on your servers.
- Keep your operating system up to date, especially security patches.
- Do not use default accounts and passwords.
- User Accounts should have very strong passwords.
- Change passwords on a regular basis and do not reuse them.
- Lock accounts after too many login failures. Often these login failures are illegitimate attempts to gain access to your system.
- Do not permit empty passwords.
- Unnecessary services should be disabled, especially remote-access unless carefully configured to control client access.
- Minimize open network ports to be only what is needed for your specific circumstances.
- Configure the system firewall. Proper setup of a firewall itself can prevent many attacks.
- Maintain server logs; mirror logs to a separate log server
- Limit user accounts to accessing only what they need. Increased access should only be on an as-needed basis.
- Maintain proper backups.
- Don't forget about physical server security.

Please refer to information about securing your specific web server and OS, which can generally be found online.

Obfuscating Definitions

You can obfuscate or "scramble" Logi Info application definition files and schedules to prevent malicious users from viewing them, protecting your work from theft, tampering, and reverse-engineering. This should be one of the final steps performed prior to deploying the application to production. Logi Studio includes features for scrambling and unscrambling definitions and they're discussed in our document *Using Logi 12 Studio*.

Logi Security

Logi Info includes features, collectively called **Logi Security**, for securely controlling access to reports and data.

The following topics provide an explanation of the principles behind Logi Security and details of its implementation:

- Authentication - Who is the User?
- Authorization - What is the User Allowed to Do?
- Adding Security to a Logi Application

A companion topic, "Working with Logi Security" on page 37, provides more detailed information for developers, and another one, "Logi Security Scenarios " on page 59, offers step-by-step guidance for implementing security in different circumstances.

About Logi Security

Logi Security provides a flexible mechanism for integrating security features into a Logi application in almost any environment. It lets you can take advantage of the User and Role/Group identification features in Microsoft Windows operating systems, including NT Security and Active Directory domain security, and in other LDAP-based and custom-built security systems.

The Logi Security implementation utilizes the traditional concepts of user *authentication* and *authorization*, which are discussed in detail in the following sections.

Security Rights can be assigned to users and then used within a Logi application to restrict users access on these levels:

- Report-level - controls access to report pages.
- Element-level - controls access to portions of reports and report components, and controls report navigation.
- Row-level - controls access to individual data rows.
- Column-level - controls access to individual data columns.

Logi Security gives you, as the developer, complete control over all aspects of report usage.

Working with Logi Add-on Modules

Logi Info Add-on Modules deliver specialized elements and complete applications to users. All are compliant with Logi Security in general and may also have unique constants or security rights that enable access to specific features. Security requirements and settings for add-on modules are discussed in detail in their configuration documents.

Working with LDAP

The Lightweight Directory Access Protocol (LDAP), used for querying hierarchical sets of records, is commonly used to store user and role information. Microsoft's **Active Directory**, and Oracle's **OpenDJ**, and Linux **OpenLDAP** are all examples of LDAP implementations.

In a typical Logi application that uses Logi Security, user credentials entered via a login page are compared against an LDAP server to authenticate the user and retrieve role information. Logi Info includes specialized elements for this purpose, for use with both .NET and Java applications.

Order of Operations

When Logi Security is enabled in a Logi application and a report is browsed, one of the very first things that happens is security processing. Therefore, it's not possible to give users the option to change security-related settings, such as the login domain, at runtime. By the time you could present users with Input controls to select their domain or choose other options, security processing has already occurred.

Working with SSL

Secure Sockets Layer (SSL) is the standard security technology for establishing an encrypted link between a browser and a web server. It ensures that all data passed between the two remains private and complete.

SSL is implemented as a web server configuration and browsers must be able to work with it. Because SSL works with the Transport layer, it does not directly interact with applications. Generally, Logi applications work well with, and are independent of, SSL implementations. No special configuration of a Logi application is required to allow it to work with SSL.

Working with Network Access Servers

Companies that implement central authentication servers using protocols such as TACACS+ and RADIUS will have no difficulty using Logi applications. These protocols, used along with network access servers, generally serve as gatekeepers for access to networks and servers at the communications level. Once access has been granted, the traffic and content from the web servers used to distribute Logi reports operate normally.

In general, security servers that maintain centralized security credentials for users *may* be accessible as a direct source of credentials for Logi applications. As discussed earlier in the publication, Logi security can work with custom SQL databases that contain security data.

However, the availability of access to these databases, encryption schemes used on the stored data, and other factors make it impossible to state unequivocally that Logi security can access centralized security data in all environments.

FIPS Compatibility

Logi Security is compatible with Federal Information Processing Standards (FIPS) security. Specifically, it will work correctly when a user's local or global security policy has its System Cryptography setting configured to "Use FIPS compliant algorithms for encryption, hashing, and signing".

Sample Security Applications

Report-, element-, and record-level security, and three different authentication schemes, can be found in *Sample App Descriptions* or on the DevNet Sample Apps page.

Authentication - Who is the User?

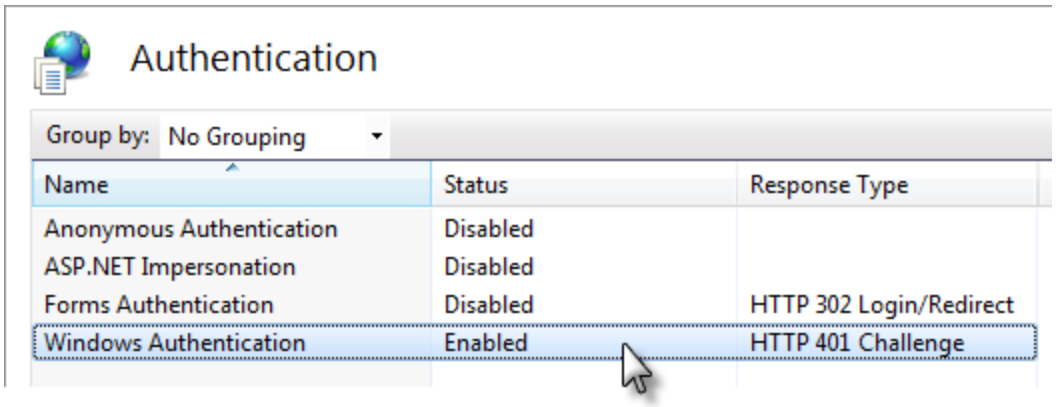
Authentication is the process of identifying the current user, usually the "login" or "logon" process that typically associates a user record with a password. For Logi applications, this can be accomplished by relying on operating system security features, by building a custom Logi-based authentication system, or by letting a parent application handle it. These three authentication methods are discussed in this topic.

Once a user is authenticated, a Logi application has access to an internal username that uniquely identifies the current user and drives the authorization process discussed later.

Operating System Authentication

This type of authentication is more commonly used in intranet/extranet environments where web server or domain administrators can easily maintain the list of valid users.

When using Windows OS-based authentication, the IIS web server manages access to a Logi application. IIS determines the user's identity and that identity gets passed into the Logi application.



IIS has a number of authentication options, configurable in the IIS Manager tool (part of which is shown above). Microsoft provides more detailed instructions for using IIS security, but generally you disable **Anonymous Authentication** and enable **Windows Authentication**, which validates the user with an account established either on the web server or elsewhere in the network domain. When necessary, the browser displays a login dialog box.

When using Java-based web servers on non-Windows operating systems, the availability of authentication via the web server can be similar but will vary by OS and web server.

Custom Authentication Systems

Custom authentication systems generally provide their own user database that can be queried to authenticate users. Logi Security provides a default login page (`rdLogon.aspx` for .NET, `rdLogon.jsp` for Java) that can be used with custom authentication systems. If authentication succeeds, custom systems must pass at least a username into the Logi application.

Custom authentication may work best when operating system-based accounts cannot be easily managed, such as when using a 3rd-party web hosting provider.

Parent Application Authentication

Non-Logi applications that call a Logi report or embed Logi components may do initial user authentication and establish a "single sign-on" relationship with the Logi resources using Logi SecureKey authentication. This technology allows the "parent" application to identify authenticated users to the Logi application/components by passing a security token. This is very useful for developers who want to extend their own applications by integrating Logi analytics and visualization technology into them.

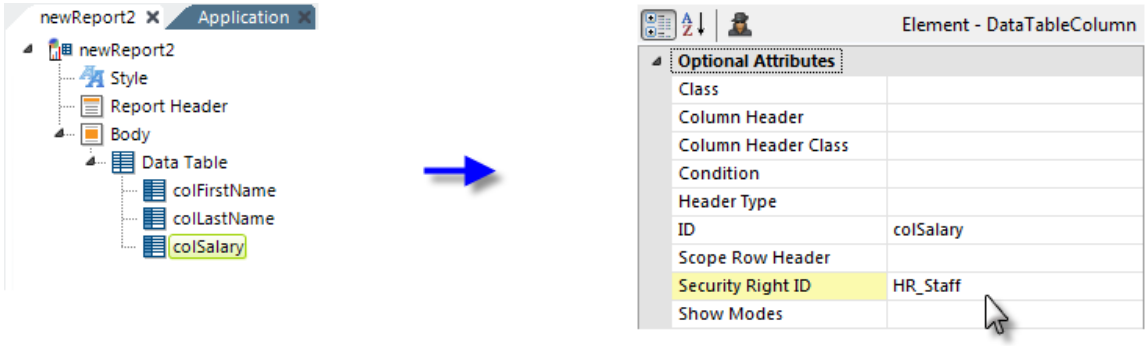
Once we know *who* the user is, we need to know what he's allowed to do. For more information on Logi SecureKey, see "SecureKey Coding Examples" on page 101.

Authorization - What is the User Allowed to Do?

Authorization determines what an authenticated user is allowed to view and what actions they can perform. Logi Info does this with Security Rights, which are granted to the user when they're authenticated.

Rights may be granted to the user individually or because he belongs to a user "Group" that has been granted a set of Rights. Logi Security uses the term "Role" instead of "Group" to designate multiple users.

When the application runs, Rights are represented internally as a string of comma-separated words. For example, "AllUsers, ChicagoStaff, Executives". When Logi Security is in use and debugging is turned on, you can view this string after the user is authenticated in the Debugger Trace Page.

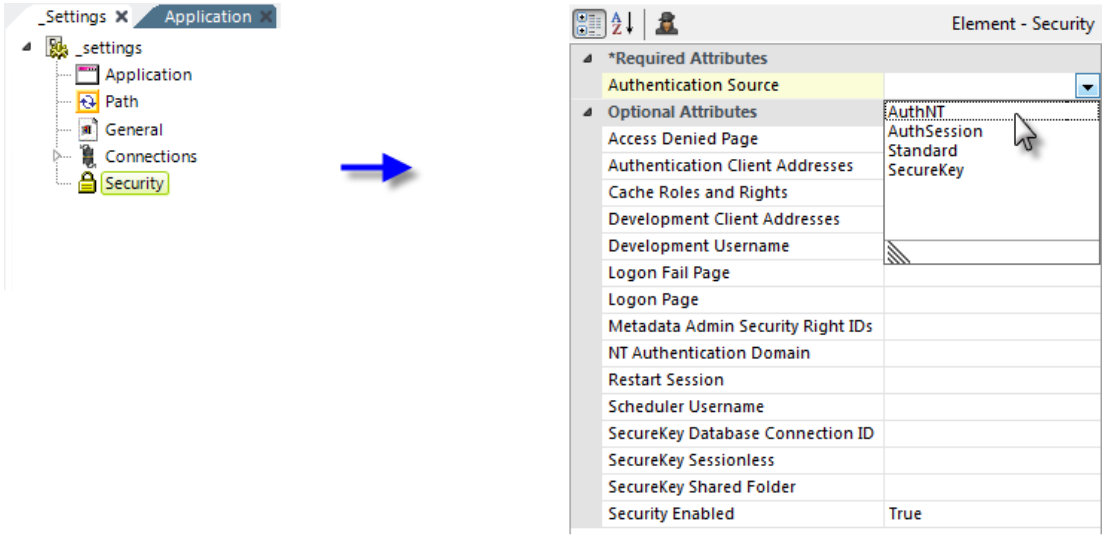


Most of the elements used to create Logi applications are "security-aware": they have a **Security Right ID** attribute that restricts their use to users with the specified Rights. The example above shows a Data Table Column element configured to only be visible to users who have the *HR_Staff* security Right.

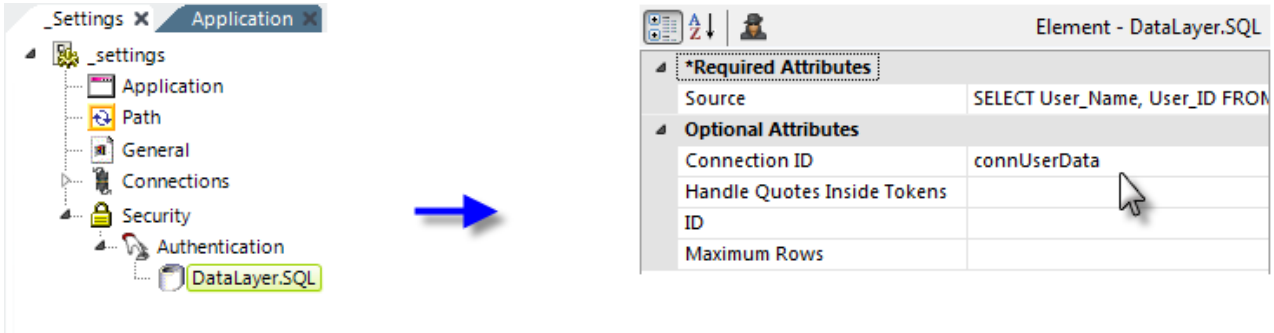
Using the earlier example, if a user had the Rights "AllUsers, ChicagoStaff, Executives" and the Security Right ID attribute value above was *Executives*, the user would be able to see the Data Table column. The attribute will also accept multiple values in a comma-separated list.

Adding Security to a Logi Application

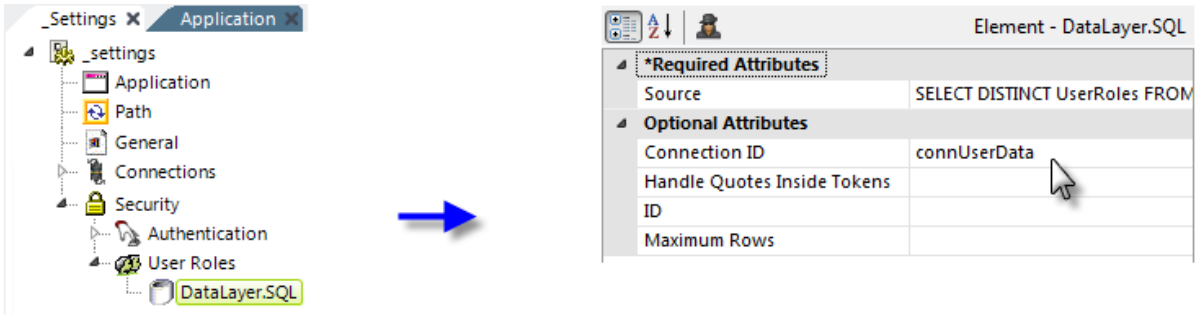
This topic briefly describes the major elements that are used in a common implementation of Logi application security.



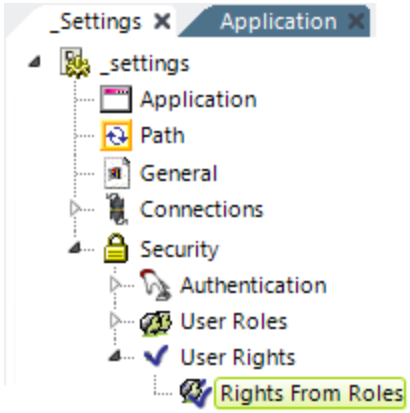
Security is enabled and globally-configured in the **_Settings** definition. As shown above, the **Security** element, which becomes the parent of all other security-related elements, has a number of important configuration attributes. These include the **Authentication Source**, which identifies the *type* of authentication in use, and **Security Enabled**, which *enables* or *disables* security altogether.



The actual authentication process is dependent on the Security element's Authentication Source attribute settings. When set to *AuthNT* or *SecureKey*, authentication takes place outside the application. When set to *Standard* or *AuthSession*, authentication takes place inside the application and an **Authentication** element with a datalayer is used, as shown above, to query a data-source to authenticate the user. To call a Process Task during authentication, add a SecurityProcess element as a child of the Security element.



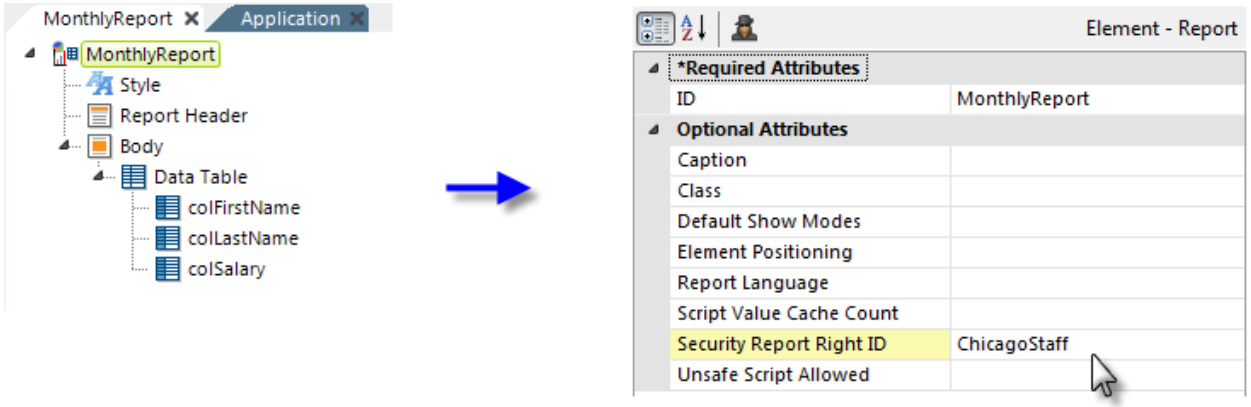
If the security scheme uses Roles (or "Groups"), the application can determine a users' Rights based on their Roles. When *AuthNT* or *SecureKey* authentication is used, the application receives the Roles automatically. When *Standard* or *AuthSession* is used, the Logi application must retrieve the roles for itself from a datasource, using the **User Roles** element with a datalayer, as shown above.



If Rights haven't been determined automatically when using *AuthNT* or *SecureKey* authentication, they need to be enumerated using a **User Rights** element. One or more of the three Rights-retrieval elements are then used:

- **Rights From Roles** - Shown above, builds the list of rights directly from the User Roles values (the Roles and Rights are the same)
- **Rights From Datalayer** - Retrieves the rights from a datasource using a datalayer
- **Right From Role** - Defines individual Rights and uses a datalayer to retrieve Roles associated with each Right.

Let's see one more example of Rights-based access restriction in practice:



In order to restrict access to an entire Logi report, its Root element's **Security Report Right ID** attribute value is set to a Right, as shown above. An "Access Denied" page is returned in lieu of the report page if a user tries to view this page but does not have the matching Right.

Working with Logi Security

Logi Security can be used to secure reports and data from being viewed by unauthorized users. This topic is for developers who already understand the underlying concepts of Logi Security and it discusses the elements and mechanisms involved in its implementation.

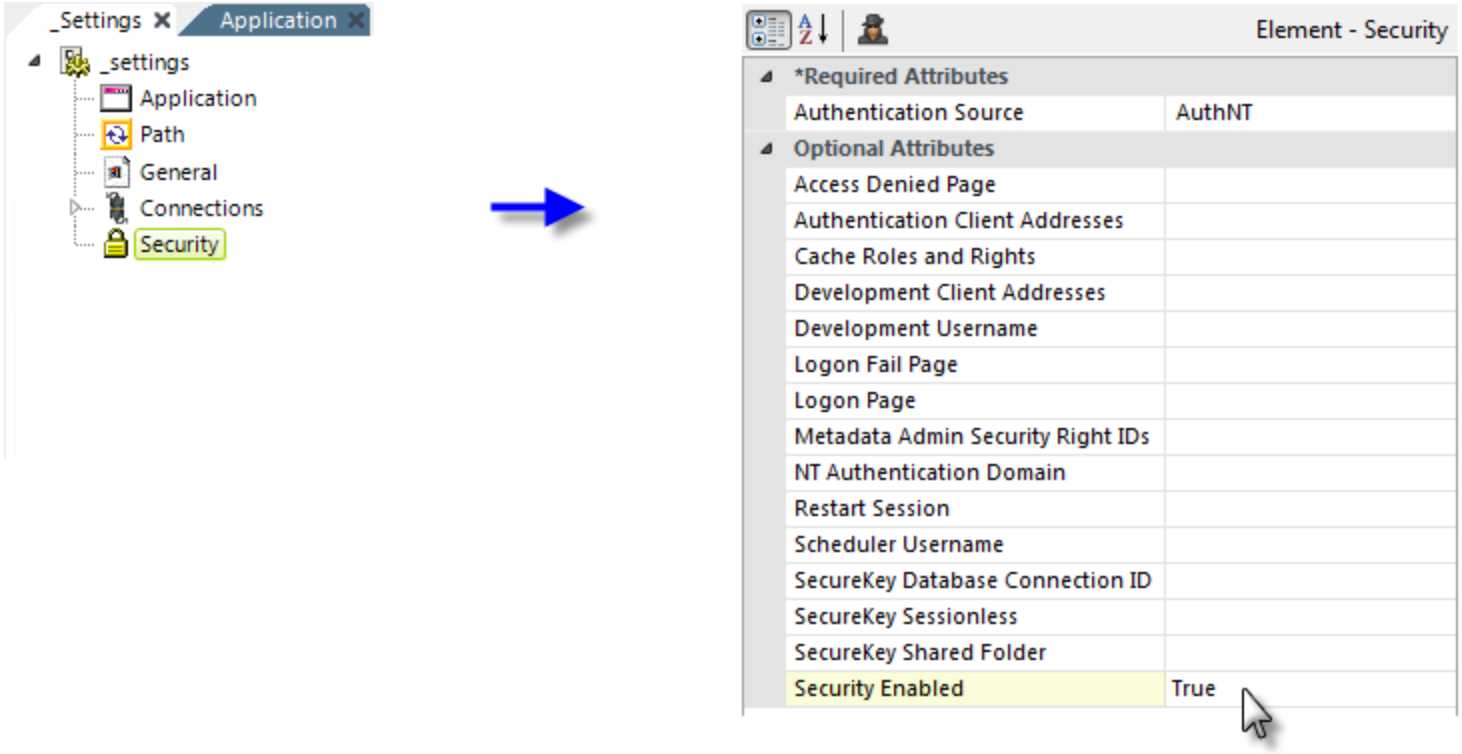
The following topics discuss the use of Logi Security:

- The Security Element
- Authenticating the User
- Gathering Role Information
- Working with Rights
- Securing Reports and Elements
- Securing Data

If you're unfamiliar with Logi Security, you should begin by reading "Logi Security" on page 25.

The Security Element

Logi Security is built around the **Security** element, which is added into a Logi application's _Settings definition. It's the container element for all other globally-configured security elements and handles both *authentication* and *authorization*.



The Security element, shown in the example above, has a **Security Enabled** attribute, which enables and disables Logi Security. The element's other attributes include:

Attribute	Description
Authentication Source	(Required) Specifies how the Logi Server gets its authentication information. Choices include: *AuthNT* - In Logi .NET applications, uses Windows OS security to get the information. This must also be enabled in the IIS web server by disabling "Anonymous Authentication" and enabling "Windows Authentication" for the application. Does not apply for Java applications. *AuthSession* - Uses a session variable, *rdUsername*, that's set in a custom logon process. *Standard* - Causes Logi application to display a form-based logon page. The supplied default logon page, *rdLogon.aspx*, or a custom page based on it, can be used. *SecureKey* - User information passed from another application, using a secure key token. For more information, see "Logi SecureKey Authentication" on page 77.
Access Denied Page	Specifies a custom page that users will be redirected to if they attempt to access a report that they're *not authorized* to access. If left blank, a standard error page distributed with Logi Info will be used. Custom error pages should be an `.aspx` file, placed in the Logi application folder.
Authentication Client Addresses	(Required if the Authentication Source is set to *SecureKey)* Specifies a comma-separated list of one or more IP addresses for approved external applications that will be requesting keys under Logi SecureKey Authentication. Computer names may be used, and IP addresses with wildcard masks. To use wildcards, specify an IP address, the space character, then the wildcard mask. For example, to allow all addresses in the range of 172.16.*.*, specify: `172.16.0.0 0.0.255.255` Generic information is available about defining IP wildcard masks.
Cache Roles and Rights	Specifies *when* a user's Roles and Rights are retrieved. Selecting *Session* causes this to occur only once, at session start. If left blank or set to *False*, Roles and Rights are retrieved with *every* request. Must be set to

Attribute	Description
	SecureKey authentication source is selected.
Development Client Addresses	*Provides special functionality for testing during development*. Specifies a comma-separated list of one or more IP addresses, with a default value of *127.0.0.1*. (IPv6 addresses, such as ::1, are also supported). If a user browses the application from one of the IP addresses in the list, then the internal security username is automatically set to the value of the Development Username attribute, and the "usual" security is bypassed.
Development Username	*Provides special functionality for testing during development*. Specifies a default user name value if a user browses the application from one of the IP addresses specified in the Development Client Addresses attribute. Let's developers avoid repetitive log-ins when testing.
Logon Fail Page	Specifies the page that users will be redirected to if logon fails. If left blank and a *Standard* authentication source is selected, a default page, *rdLogon.aspx* is used. A custom page named here should be an `.aspx` file, placed in the Logi application folder.
Logon Page	Specifies the page displayed at logon when a *Standard* authentication source is selected. If left blank, a default page, *rdLogon.aspx*, is used. Custom logon pages can be created by copying and renaming the default logon page and placing them in the Logi application folder.
Metadata Admin Security Right IDs	Specifies a comma-separated list of Security Right IDs that will be granted access to the Web Metadata Builder, see Web Metadata Builder. The default value is *rdMetadataAdmin*. To grant access to the Web Metadata Builder, either give users the rdMetadataAdmin security Right, or set this attribute to an existing administrator security Right.

Attribute	Description
NT Authentication Domain	Specifies the Windows domain that will be authenticating users when an *AuthNT* authentication source is selected. The application will then *only* accept users authenticated in that domain. To authenticate users from multiple domains, leave this attribute value blank.
Restart Session	When using a *SecureKey* authentication source and set to *True*, this attribute causes almost all session variables to be cleared at the start of each attempted login (the exceptions are several Logi system variables). This feature allows a single browser instance to access the same Logi application and use different credentials and sessions from different browser tabs. Default value: *False.*
Scheduler Username	When using an *AuthSession* or *Standard* authentication source, specifies the name of the user account that will be used to run scheduled Logi processes.
SecureKey Database Connection ID	Specifies a database Connection element ID, enabling the temporary storage of SecureKeys in a relational database rather than in the file system. The temporary values are stored in a database table named *rdSecureKey*, which is automatically created the first time a SecureKey is used. Supported databases are: • SQL Server • Oracle • MySQL • PostgreSQL
SecureKey Sessionless	When using a SecureKey authentication source and set to *True*, this attribute causes SecureKeys to be stored in the application data cache folder (by default, rdDataCache) instead of in Session state. These files expire and get cleaned-up automatically if no requests for them are made during the Session timeout period. Setting this attribute to True is especially important when securing access to Data definitions to

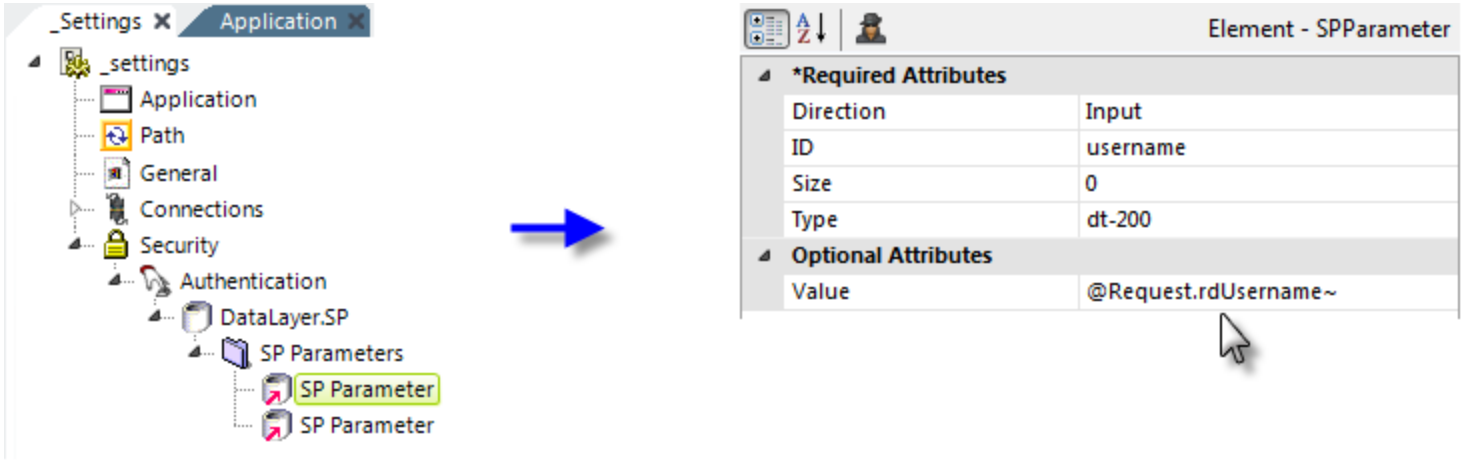
Attribute	Description
	retrieve JSON data because such requests are sessionless.
SecureKey Shared Folder	This attribute is functional only when using a SecureKey authentication source and clustered web farm or web garden configurations. In a single-server configuration, SecureKey keeps SecureKey requests in Application state. With multiple servers, this information must be stored in files in the folder specified by this attribute, which is shared among the web servers. The account used by (or impersonated by) the web application must have network access rights to read, write and delete files from this folder. Set this attribute value to UNC network path, such as: `//mySharedServer/SecureKeyFolder`Older files in the SecureKeyFolder are automatically deleted over time, so do not use this folder to store other files.
Security Enabled	Enables or disables Logi Security. The default value is *False*.

Bypassing Security During Development

Adding security to a Logi application complicates the development process, as security will be applied when you Preview a report in Studio or run a wizard. In order to make life a little easier for developers, two special Security element attributes can be used to disable, or apply special, security settings while developing and testing an application. These are the **Development Client Address** and **Development Username** attributes, described in the table above.

Authenticating the User

The first step in providing secure access to the Logi application is to authenticate the user, based on the login. The Security element's **Authentication Source** attribute value governs the behavior of some of the other security-related elements for this step. When this attribute is set to *AuthNT* or *SecureKey*, the user is authenticated outside of the Logi application and the token `@Function.UserName~` is set automatically.



When the attribute is set to *AuthSession* or *Standard*, it may be the Logi application's responsibility to authenticate the user by querying a datasource. An **Authentication** element, with a child datalayer, as shown above, is used for this purpose. The example above uses a DataLayer.SP element and its child elements to call an SQL database Stored Procedure (SP). The first SP Parameter element is configured, as shown above, to pass the login user name, and the second is configured for the login password, using the token `@Request.rdPassword~`.

The two case-sensitive token names, `rdUsername` and `rdPassword`, match the IDs of text input controls used in the standard Login page provided with each Logi application, and should also be used as the input control IDs in any *custom* Login page you might create. The authenticating stored procedure code should be similar to:

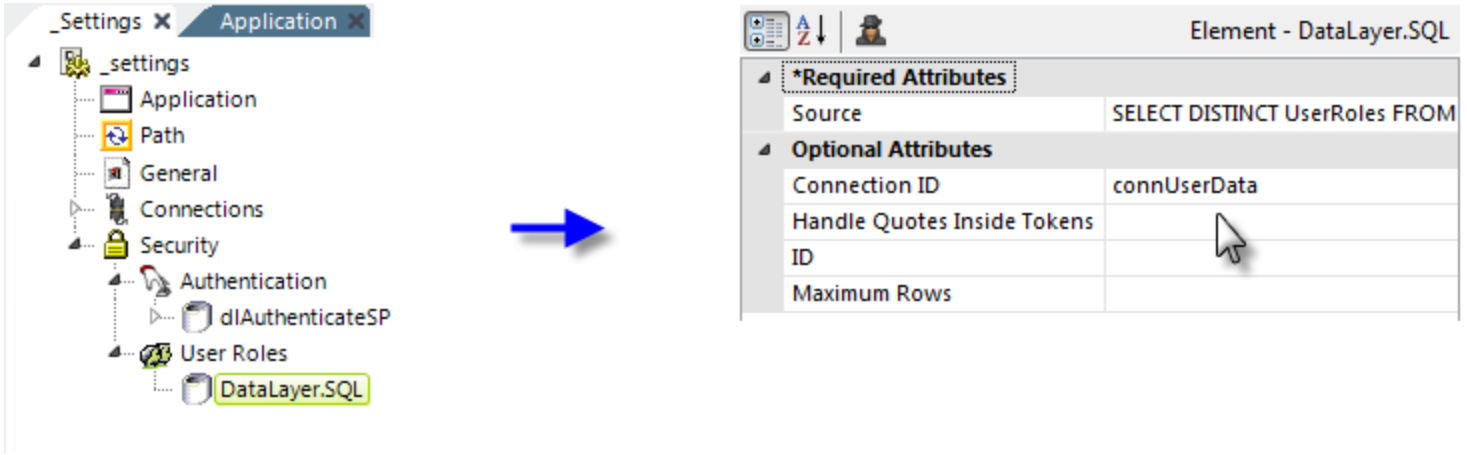
```
@username varchar(20),
@passwordvarchar(16)
SELECT User_Name, User_ID FROM UsersTable
WHERE User_LoginID = LTRIM(@username) AND User_Password = LTRIM(@password)
```

Parameters are passed to the SP in the *top-to-bottom order* of the SP Parameter elements and do not take the parameter IDs into account. Your SP should "receive" and define the parameters in the same order. If the SP returns *no* data, then the user is *not* authenticated. A successful authentication will return two values: the first one will be automatically assigned internally to the token `@Function.UserName~` and the second to `@Function.UserID~`. For additional flexibility, the Authentication element includes an attribute, **Username Data Column**, that lets you specify the name of the column in the result set that contains the user name.

Different datalayer elements can be used with the Authentication element, so that users can be authenticated using a variety of datasources. However, we *do not* recommend using **DataLayer.SQL** with direct query text replacement using tokens; it's a possible avenue for "SQL Injection" attacks.

Gathering Role Information

The gathering of Role information, if used, is dependent on the Security element's **Authentication Source** attribute settings. When it's set to *AuthNT*, Roles are automatically defined based on the NT Security or Active Domain group information from the OS. The actual Roles are maintained internally as string values, such as "Executive" or "EndUser"; multiple Roles are represented as a comma-delimited string of values.



When the Authentication Source attribute is set to *Standard* or *AuthSession*, the Logi application must retrieve any Role information for itself from a datasource, using the **User Roles** element with a datalayer, as shown above. When Authentication Source is set to *SecureKey*, there are two choices: the list of Roles can be included in the communication from the Parent application and will be received into the User Roles element directly, or they can be retrieved from other sources by using User Roles with a datalayer. In cases where a datalayer is used beneath the User Roles element, by default Roles are expected to be returned either as values in the first column in one or more records, or as multiple values in a comma-delimited list in the first column of the first row. Example query:

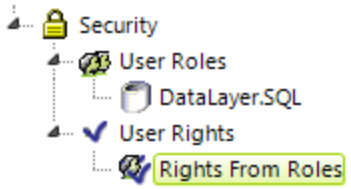
```
SELECT Roles FROM myUserTable WHERE UserName = '@Function.UserName~'
```

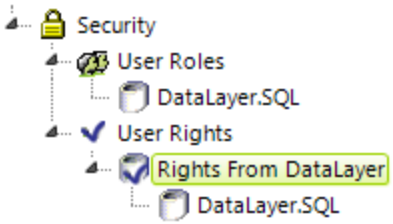
For additional flexibility, the User Roles element includes an attribute, **Roles Data Column**, that lets you specify the name of the column in the result set that contains the role information.

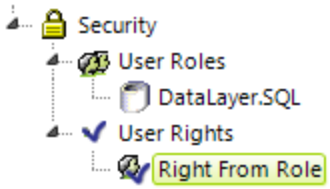
 The tokens `@Function.UserRights~` and `@Function.UserRoles~` are *not* populated until processing of the Security element is complete. Therefore, you cannot use them in a SQL query like the one shown above. The `@Function.UserName~` token is available immediately after the login page is submitted, so it can be used. As with all datalayers, the results retrieved can be affected by adding **Calculated Column** and/or **Filter** elements beneath them. See for information about how user Roles are included in SecureKey communications. In all cases, the end result is that the Roles for the current user are maintained as a list that can be referenced internally by the Logi application.

Working with Rights

You'll only need to deliberately retrieve user Rights when the Security element's Authentication Source attribute is set to *Standard* or *AuthSession.* The Rights are maintained internally as string values, such as "ExecutiveReport" or "HRStaff"; multiple Rights are represented as a comma-delimited string of values. Rights are retrievedusing a **User Rights** elements as a container and then one or more of these three rights retrieval elements:





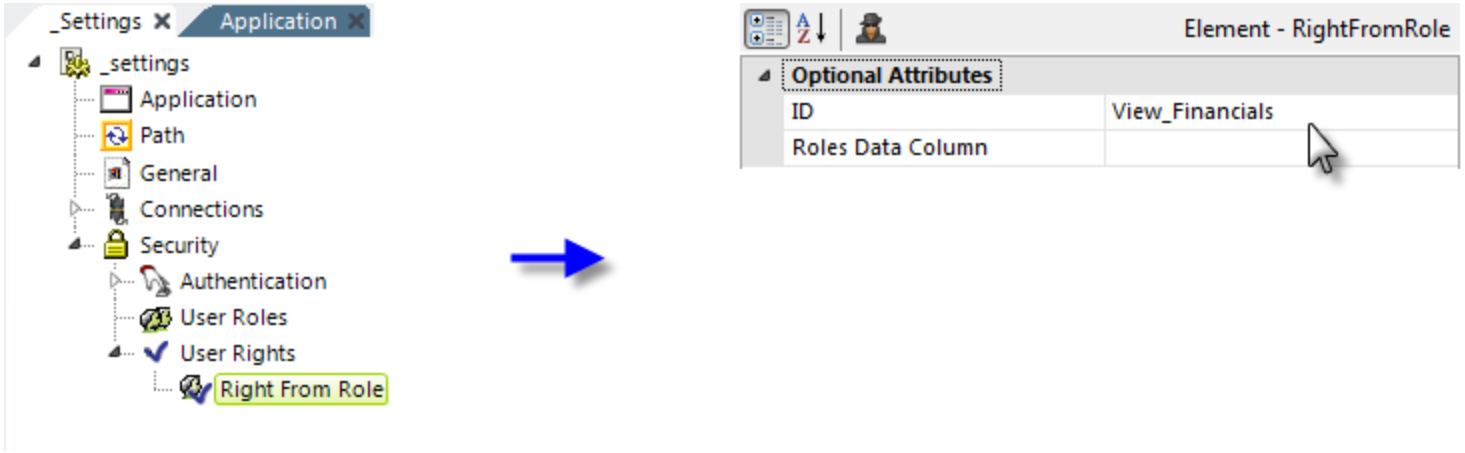


The examples shown above include a User Roles element, but it's not necessary if Roles are supplied to the Security element via the SecureKey request. The separate techniques and elements used to retrieve the Rights are described below:

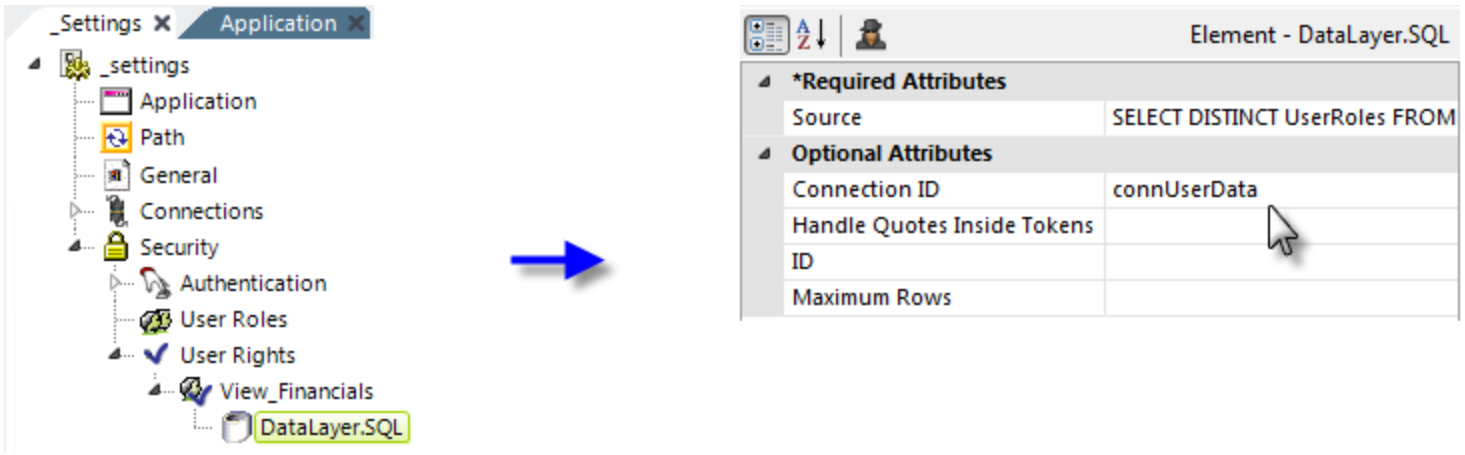
Element	Technique
Rights From Roles	Builds the list of Rights directly from the Roles values. The Roles and Rights are the same.
Rights From DataLayer	Builds the list of Rights directly from its child datalayer. The data returned should include a Right in the first column of each row, or multiple Rights in a single row as a comma-separated list.
Right From Role	Defines each right individually, with one Right From Role element for each right. The element's ID is the Right value, which corresponds to Security Right ID attributes in elements in report definitions. Right From Role

Element	Technique
	works with user Roles: if a user has a Role that is returned by this element's child datalayer, then he is granted the Right specified by the element's ID. The child datalayer should retrieve data with a Role value in the first column of each row, or multiple roles in a single row in a comma-separated list.

These elements may be used in any combination; the Rights retrieved by each are combined together into a comprehensive Rights list. The following example shows the use of a **Right from Role** element to define a Right:



In the example above, the element's **ID** attribute provides the name of the Right being defined.



Then a datalayer is used to determine if the user has a specific Role. A typical query might be:

```
SELECT DISTINCT UserRoles FROM myUserTable WHERE Role = 'Executive'
```

If the user has a Role in his Roles list that matches a Role returned into the datalayer using this query, then the user is accorded the Right specified by the Right From Role element ID. In other words, in this example, a user with the Role of "Executive" would be given the Right "View_Financials". Complex combinations of rights are possible: The Logi engine applies OR logic when processing a comma-delimited list of user rights and will allow a definition or element to be processed or displayed if the user possesses at least one of the rights in the list. If, instead, you want to apply AND logic processing, you can string user rights together using the pipe ("|") character and the engine will consider them to be one right. For example, rights "A|B|C, D|E|F" mean that the user must possess either A and B and C rights, or D and E and F rights; they will not be permitted access if they only possess A or D.

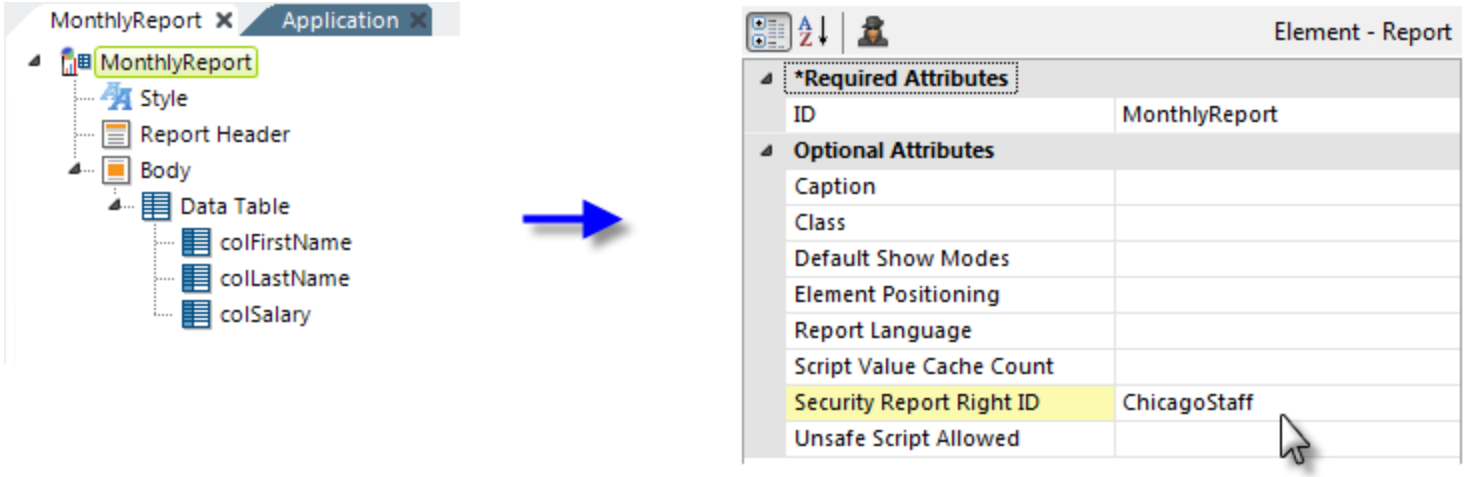
The tokens `@Function.UserRights~` and `@Function.UserRoles~` are *not* populated until processing of the Security element is complete. Therefore, you cannot use them in a SQL query like the one shown above.

Securing Reports and Elements

Logi Info, using Logi Security, prevents unauthorized users from viewing reports and from seeing or using individual elements in reports.

Securing Reports

Securing an *entire report* is easily accomplished by assigning one or more Rights to the report; it will not be generated for unauthorized users.

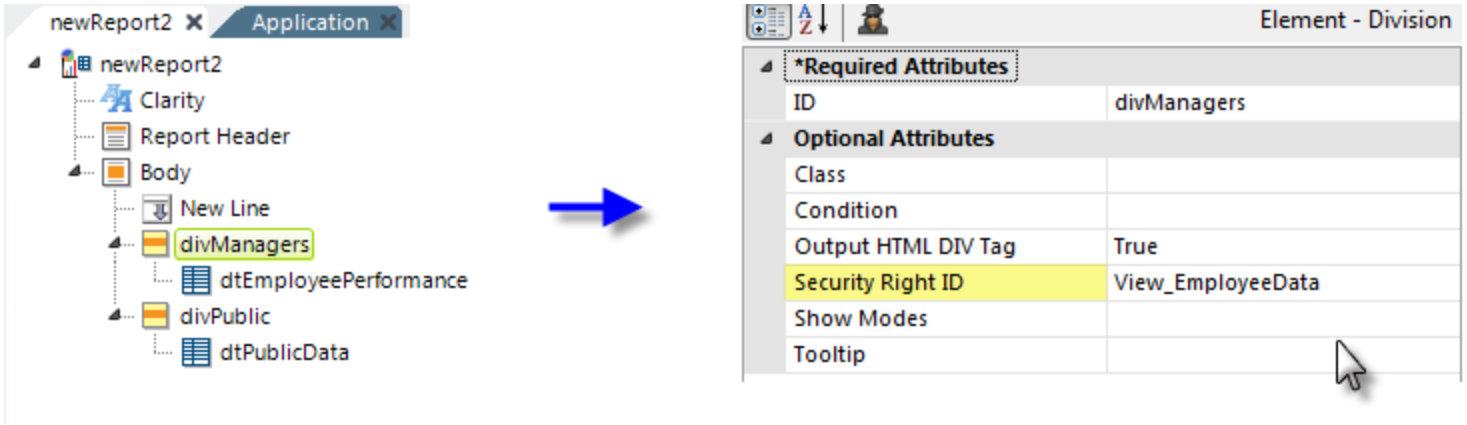


In order to restrict access to an entire Logi report, its Root element's **Security Report Right ID**attribute value is set to a Right, as shown above. An "Access Denied" page is returned in lieu of the report page if a user tries to view this page but does not have the matching Right.

Use caution when entering Security Right ID values: Rights are *case-sensitive*. Users who have not been accorded the identified right and who attempt to access this report will receive be redirected to the "Access Denied" page (based on the Security element's attribute of the same name), if defined, or to an error message indicating that the user doesn't sufficient permissions.

Securing Elements

In Logi Info, more than 70 elements have a Security Right ID attribute, including all of the Action, Input, Chart, and Table elements. All of them can be secured using their Security Right ID attribute.



In the example shown above, a report's Data Tables are contained within two **Division** elements. Entering a previously-defined Right in the Security Right ID attribute value for the first division and leaving that attribute *blank* for the second division causes two different versions of the report to be dynamically-generated at runtime, depending on the current user's Rights. Users who have the "View_EmployeeData" Right will see both divisions and both Data Tables; everyone else will only see the second division and table.

In general, elements secured in this manner are not even generated in the HTML page at runtime for unauthorized users. You can enter multiple values in the Security Right ID attribute, separated by commas. In that case, the user will see the restricted division if he has been granted any one of the listed Rights. This example illustrates how Logi Security can help you reduce the overall number of report definitions you have to manage and maintain through dynamic report configuration at runtime, based on user Rights.

Securing Data

It's equally important to be able to control access to data and Logi Info provides several ways to accomplish that.

Securing Connections and Metadata

The **Metadata** element provides a method of controlling the data objects available to users who use the Active Query Builder (with the Analysis Grid or InfoGo application) through the use of metadata files. The Metadata element includes a Security Right ID attribute that can be used to prevent unauthorized users from working with that data. A browser-based tool, the Web Metadata Builder, that can be used to create data connections and metadata files is available inside InfoGo and other Logi applications. Access to this tool is controlled by Security Right IDs entered into the **Security** element's **Metadata Admin Security Right IDs** attribute.

Security Tokens and SQL Queries

The user-related security information collected when the user is authenticated is available in definitions through the use of tokens, including `@Function.UserName~`, `@Function.UserID~`, `@Function.UserRoles~`, and `@Function.UserRights~`. The last two tokens provide their values, if there's more than one, as a comma-separated list.

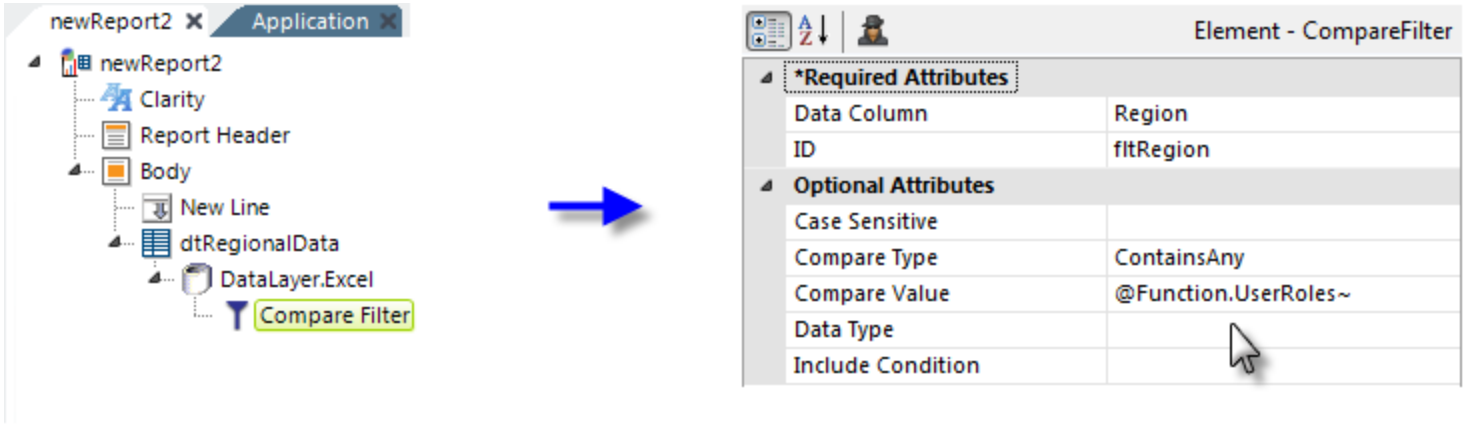
The availability of these tokens allows developers to use SQL queries and datalayer security filters to restrict data based on the user's Rights. Assume that the current user has been assigned two Roles, "Accounting" and "Retail Division" (based on the organizational departments he belongs to) and needs to view a report that summarizes expenses by department. The report should *only* present data to him for the departments he belongs to. This can be accomplished with this SQL query:

```
SELECT * FROM ExpenseData WHERE Department IN (@SingleQuote.Function.UserRoles~)
```

The `@Function.UserRoles~` token contains a comma-separated list of the user's two Roles. The inclusion of the **SingleQuote** token prefix causes each of the individual Roles in the list to be surrounded with single quotes, which is the format required for the SQL query's "IN" clause. The result set will only include rows with data for the user's two departments.

Security Tokens and Filters

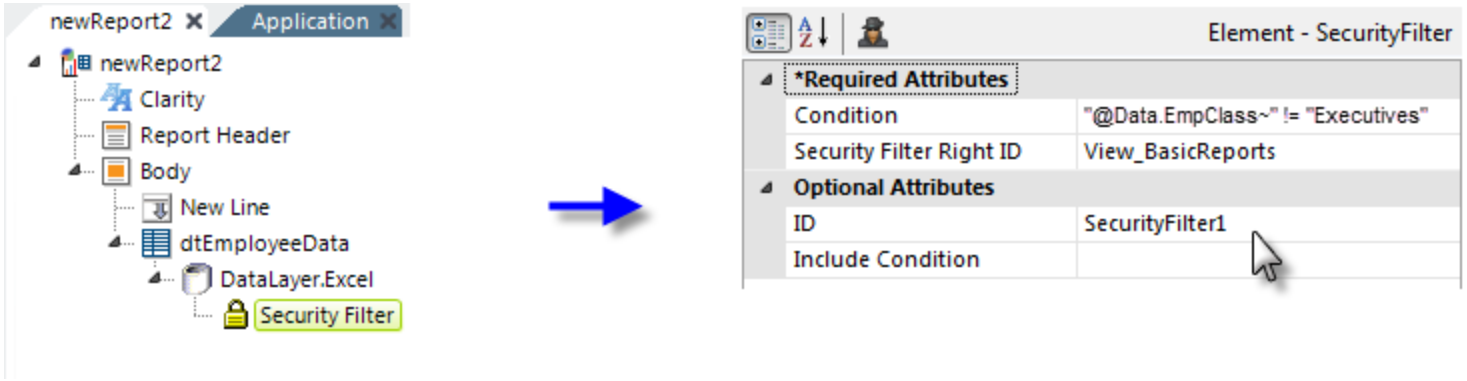
The same kind of comparisons can be used to filter data *after* it's been retrieved into a datalayer. In this example, assume users have been assigned a Role that indicates their office region, such as "Northeast" or "Midwest", and they are only allowed to see report data for their region.



As shown above, the report definition can contain a **Compare Filter** beneath its datalayer. It will filter out any rows in the datalayer whose Region column value doesn't match any of the regions specified in the user's Rights list.

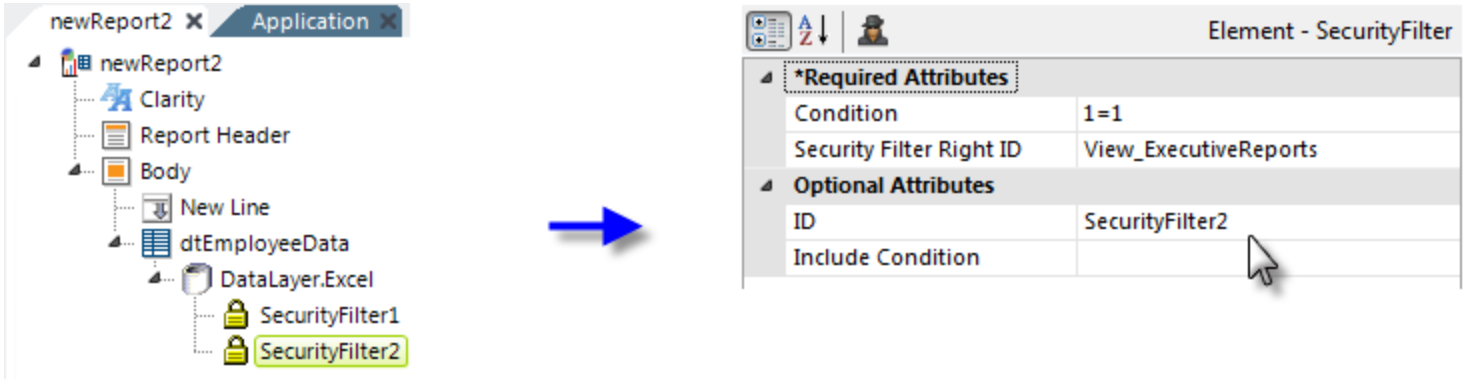
The Security Filter Element

Logi Info also includes the ability to impose *row-level* security using the **Security Filter** element, which filters data after it has been retrieved into a datalayer. To restrict access to specific data rows, one or more Security Filter elements are added as child elements of a datalayer.



In the example above, a Security Filter element has been added to a report definition that displays employee data. The purpose of this filter element is to allow basic users to see all data rows retrieved into the datalayer *except* those of Executive-class employees. Datalayer rows that *do not* cause the formula in the filter's **Condition** attribute to evaluate to *True* will be removed. Its **Security Filter Right ID** is set to the Right associated with the role of the basic user.

When the report is generated, this filter element will only be applied if the current user has the Role of a basic user.



Next, a second Security Filter element has been added, as shown above. The purpose of this filter element is to allow privileged users to see *all* data rows retrieved into the datalayer. Its Condition attribute value, which is required, is set to an expression, *1=1*, that will always evaluate to *True.* Its Security Filter Right ID is set to the Right associated with the Role of privileged users. When the report is generated, this filter element will only be applied if the current user has the Role of those allow to view Executive Reports, i.e. the privileged user.

Keep in mind that the Security Filter element behaves very differently than a Compare Filter element. Whereas the Compare Filter will remove datalayer rows that fail to meet certain criteria but allow *all others* to be available, the Security Filter prevents the datalayer from providing *any rows*, unless the current user's Rights match at least one of the Security Filter's Rights. This prevents a user who has not been granted *any* Rights from seeing all datalayer rows.

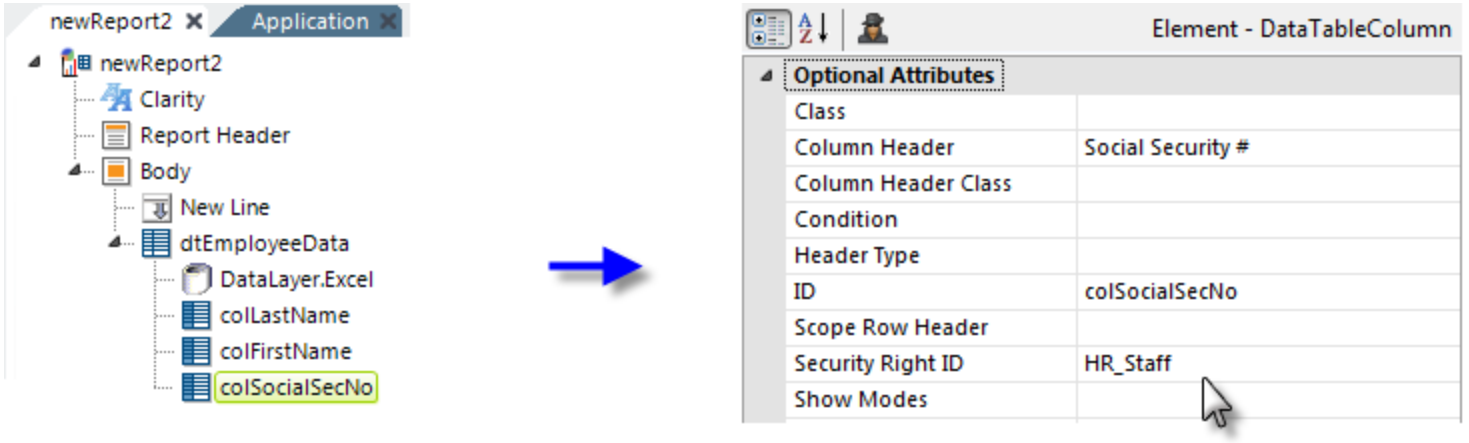
The Include Condition Attribute

The Security Filter element also has an **Include Condition** attribute. If the value of this attribute is left blank or contains a formula that evaluates to *True*, the Security Filter is applied to the datalayer. If the value evaluates to *False*, the filter is ignored and

does not affect the datalayer. This powerful feature allows you to dynamically determine if security will be applied to the datalayer or not, or to switch between different Security Filters at runtime.

Column-Level Security in Data Tables

Logi Info can also restrict data displayed in Data Tables at the *column* level by adding or removing table columns based on security Rights.



As shown above, a **Data Table Column** element can have its **Security Right ID** attribute value set so that the column will only appear in the table for users granted the required Right.

As discussed here, **Logi Security** provides opportunities to control the retrieval and presentation of data in Logi applications in a variety of ways. Developers are encouraged to experiment to find the approach best suited to their requirements.

Logi Security Scenarios

This topic introduces examples of representative security scenarios, from simple to complex, to assist you in understanding how to configure your web server and Logi applications to take advantage of Logi Security.

The scenarios include:

- IIS: Basic and Digest Authentication
- IIS: Windows Authentication
- IIS: ASP.NET Impersonation
- Logi Security: NT Authentication
- Logi Security: Session Variable Authentication
- Logi Security: Standard Authentication
- Logi Security: SecureKey Authentication

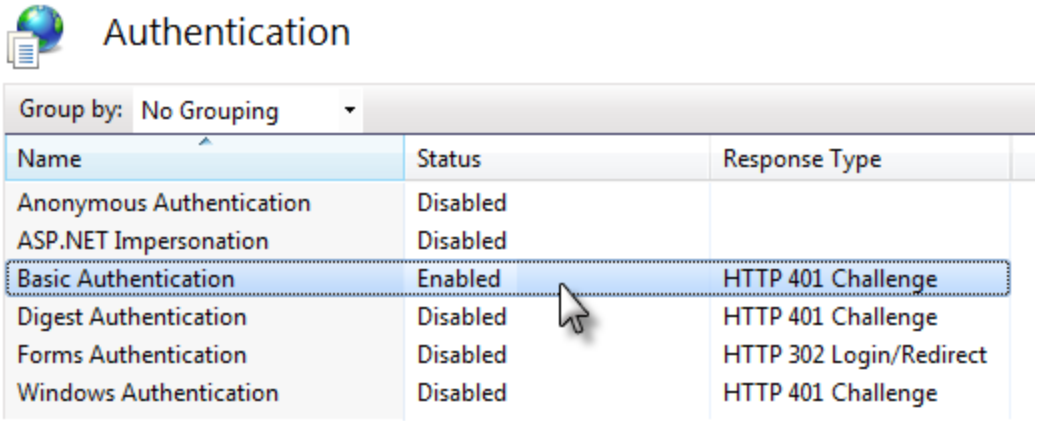
For information about Logi Security in general, see "Logi Security" on page 25 and "Working with Logi Security" on page 37. The examples use Windows IIS server, however, many of them are also relevant when using non-Windows web servers, such as Apache Tomcat. The availability of different forms of security in the IIS web server are predicated on the installation of related IIS features, accomplished through Control Panel → Programs and Features → Windows Features.

IIS: Basic and Digest Authentication

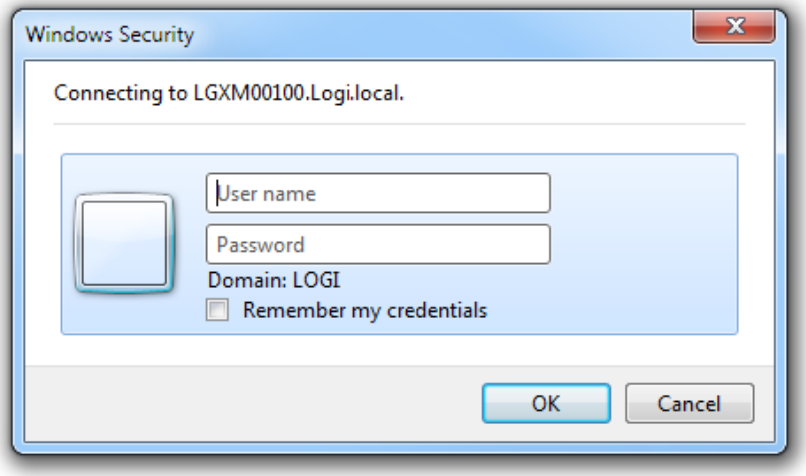
This scenario simply configures the web server, IIS 7.5, to require valid credentials before allowing any access to the Logi application.

When using **BasicAuthentication**, user login credentials are sent to the web server in plain text, without any encryption. Anyone attempting to compromise your system security could intercept network messages and examine the unencrypted credentials. This security method is *not* recommended for use on public networks. **DigestAuthentication** improves on this by sending hashed credentials.

To configure your Logi Application for either type of authentication:



1. Using the **IIS Manager** utility, select your Logi application, and then select the **Authentication** feature.
2. Disable the *Anonymous Authentication* option, as shown above.
3. Enable the *Basic Authentication* or *Digest Authentication* option, as shown above.
4. Exit the feature and restart the appropriate Application Pool.

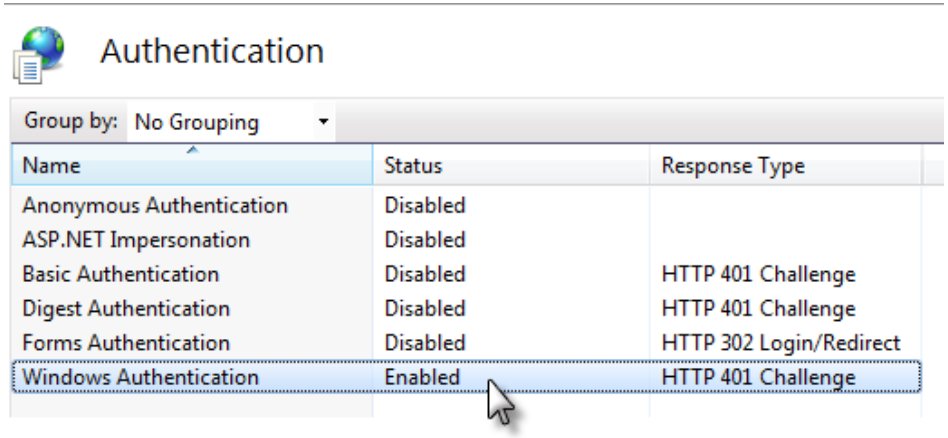


Now, when your Logi application is browsed, users will see the dialog box shown above and will have to provide valid credentials for your network domain before seeing the Logi application.

IIS: Windows Authentication

In this scenario, shown with IIS 7.5, the name of a user who has already been authenticated by Windows domainor network security is supplied to the Logi application. This is a very secure form of authentication and convenient for users because theydo not have to login twice.

To configure your Logi Application for either type of authentication:



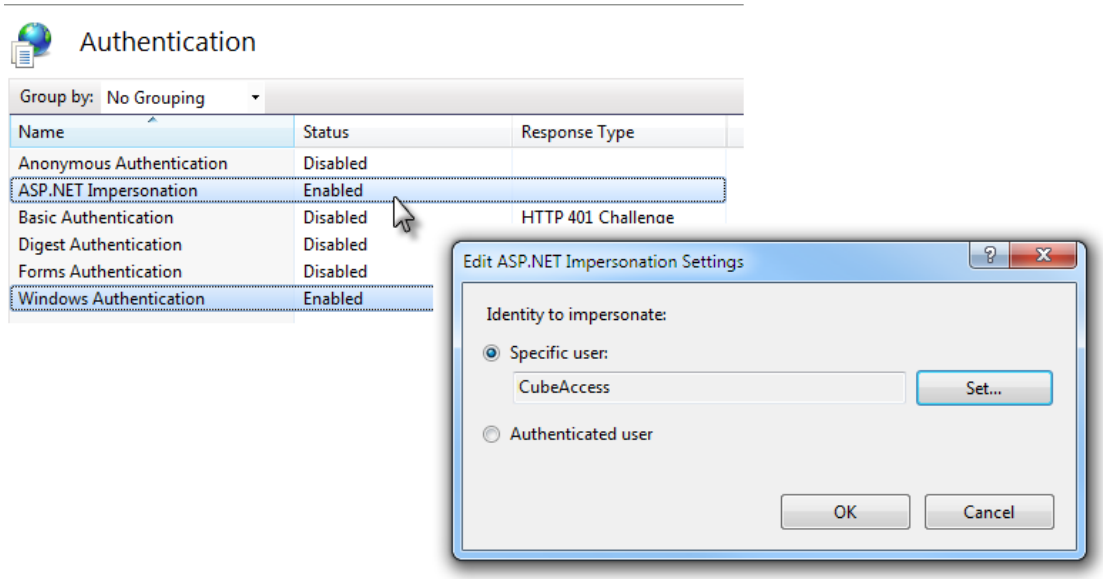
1. Using the **IIS Manager** utility, select your Logi application, and then select the **Authentication** feature.
2. Disable the *Anonymous Authentication* option, as shown above.
3. Enable the *Windows Authentication* option, as shown above.
4. Exit the feature and restart the appropriate Application Pool.

Now, when the URL for your Logi application is browsed, authorized domain users will not see a login dialog box at all; instead they'll proceed directly into the Logi application. Users who are not authorized will be redirected to an error page.

IIS: ASP.NET Impersonation

This scenario, shown with IIS 7.5, lets you run your Logi application under a different security context. This is a required configuration when using Logi OLAP to connect to Microsoft SQL Server Analysis Services cubes via the MSOLAP provider.

To configure your Logi Application for this type of authentication:



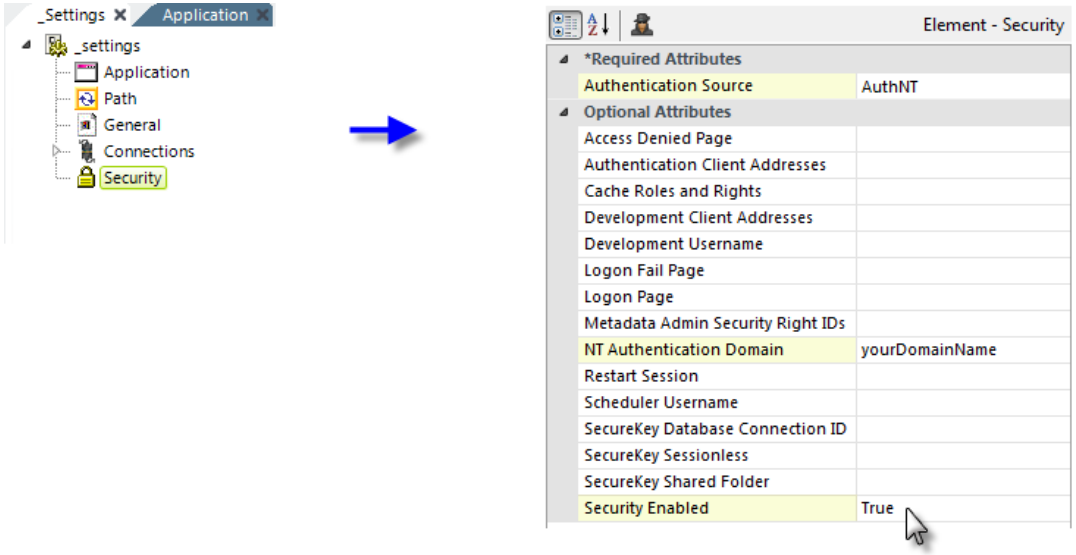
1. Using the **IIS Manager** utility, select your Logi application, and then select the **Authentication** feature.
2. Disable the *Anonymous Authentication* option, as shown above.
3. Enable the *ASP.NET Impersonation* option, as shown above.
4. Right-click the option and select Edit..., then enter credentials for the account you want to impersonate in the resulting dialog box.
5. Enable the *Windows Authentication* option, as shown above.
6. Exit the feature and restart the appropriate Application Pool.

Now, when the URL for your Logi application is browsed, authorized domain users will not see a login dialog box at all; instead they'll proceed directly into the Logi application and the appropriate impersonated credentials will be used for connection to other services.

Logi Security: NT Authentication

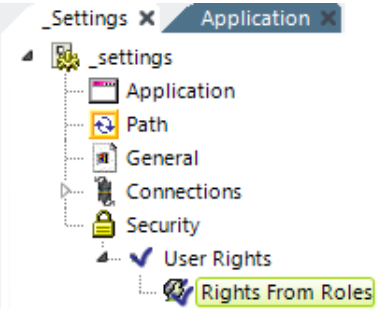
The **Security** element in the _Setting definition can be configured to use "NT Authentication" as its authentication source. This mode uses the Windows operating system's security scheme as the source of the user name and security rights and roles. This typically means the user will be authenticated in a Windows Active Directory domain but this mode can also be used when authenticating to other LDAP directories, using these special elements: **DataLayer.LDAP** and **DataLayer.LDAP Authentication**.

Assuming that your IIS web server has been configured to use *Windows Authentication* as described in "IIS: ASP.NET Impersonation " on page 63, here's how to configure your Security element:



1. In your Logi application's _Settings definition, add a **Security** element, as shown above.
2. Set the **Authentication Source** attribute value to *AuthNT*.
3. Set the **NT Authentication Domain** attribute value to the name of your domain. This value is not case-sensitive and you should not enter any slashes, backslashes, or ".com" here. If authenticating across multiple domains, leave this value blank.

4. Set the **Security Enabled** attribute to *True*.



5. The **Security** element will automatically get the Roles ("Group") information for the authenticated user from the domain, without the need for any other child elements.

   If you want to make use of user Rights, several elements are available for retrieving rights. The easiest way to do this is to add, beneath the Security element, a **User Rights** element and, beneath it, a **Rights From Roles** element, as shown above. This will cause the internal Roles list retrieved by the Security element to be copied to the internal Rights list. For other ways to work with Rights, see "Working with Logi Security" on page 37.

Save the application and browse the report. You should be able to access the report and, within the Logi application, work with the user information via tokens: the user login ID will be available as `@Function.UserName~`, user Roles as `@Function.UserRoles~`, and user Rights as `@Function.UserRights~`.

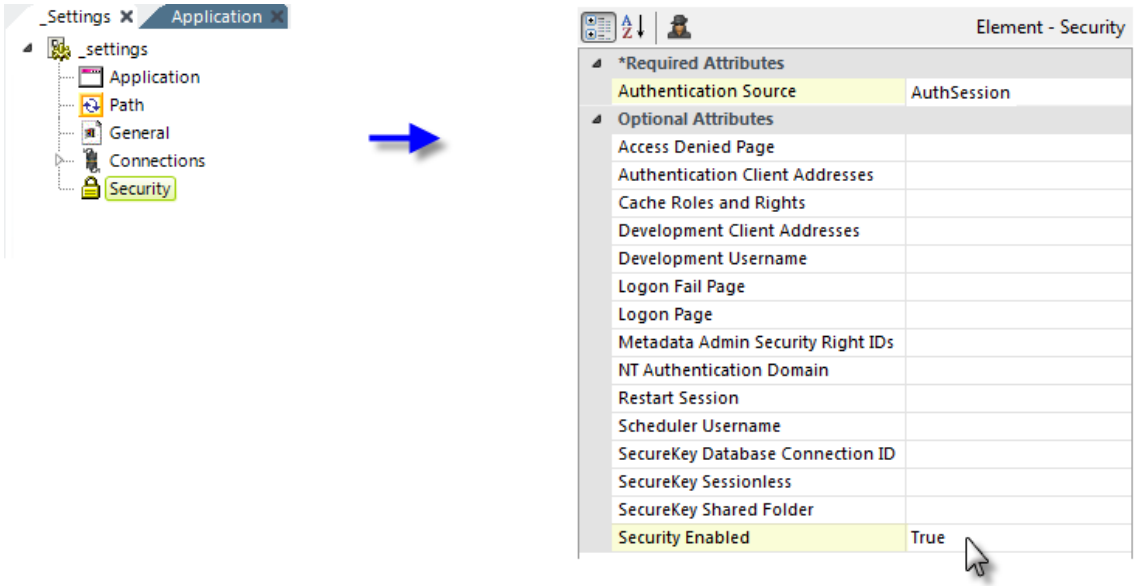
Windows and other security domains may allow usernames and passwords to contain special characters. However, many of the connection strings used to connect to datasources have reserved characters, usually single- and double-quotes, the equals sign, semi-colons, and commas. When a domain username or password containing these reserved characters is automatically

combined into the connection string, they break it. They can also affect queries, for the same reason. Therefore, domain usernames or passwords that include these reserved characters will cause a failure to connect or an outright error.

Logi Security: Session Variable Authentication

In this scenario, which uses your own custom login application or process, on successful authentication you create a Session variable, named "rdUsername", to hold the user login ID value. Logi Security can then use that Session variable to authorize access to features in your Logi application.

To do this, first configure your web server to use whatever authentication method is appropriate for your custom login application or process. Then configure your Logi Application for this type of authentication:

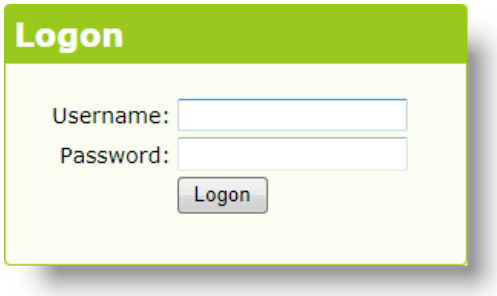


1. In your Logi application's _Settings definition, add a **Security** element, as shown above.
2. Set the **Authentication Source** attribute value to *AuthSession*.
3. Set the **Security Enabled** attribute to *True*.
4. Add additional security-related elements (Roles, Rights, SecurityProcess, etc. - not shown) as necessary.

When you login and run your Logi application, you should be able to access your login ID using `@Session.rdUsername~`.

Logi Security: Standard Authentication

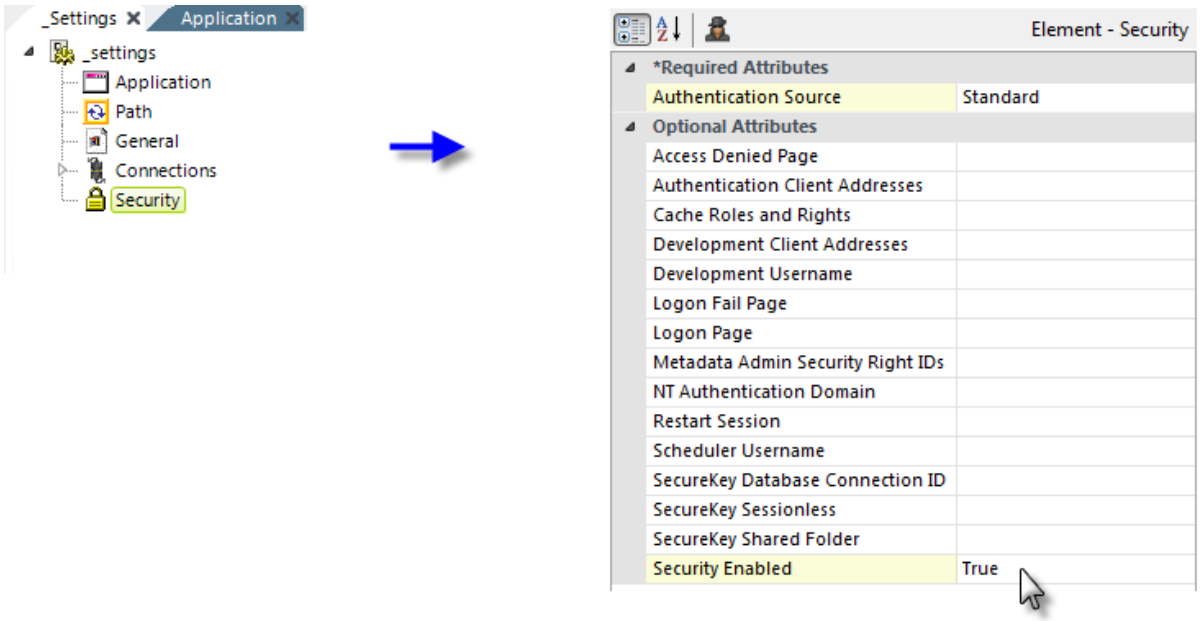
In this scenario, user data is stored in your own custom database and your Logi application validates the user against it.



Logi Info includes a default login page that includes the login form shown above (its appearance will vary slightly depending on the Theme used). This is the file `rdLogon.aspx`, in your Logi application root folder. You can substitute your own custom login page, as long as you submit user input controls with the *same names* as those in the default page.

After the login page is submitted, your Logi application handles validating the user against your security database. If the user is authenticated, the query should return one row of data and processing continues; if the user is not authenticated, no data should be returned and an error page will be displayed. A default "Access Denied" page is supplied but you can customize that, too, if desired.

To use this type of authentication with IIS, your application's Authentication feature should be configured for *Anonymous Authentication*. Then, to configure your Logi Application for this type of authentication:

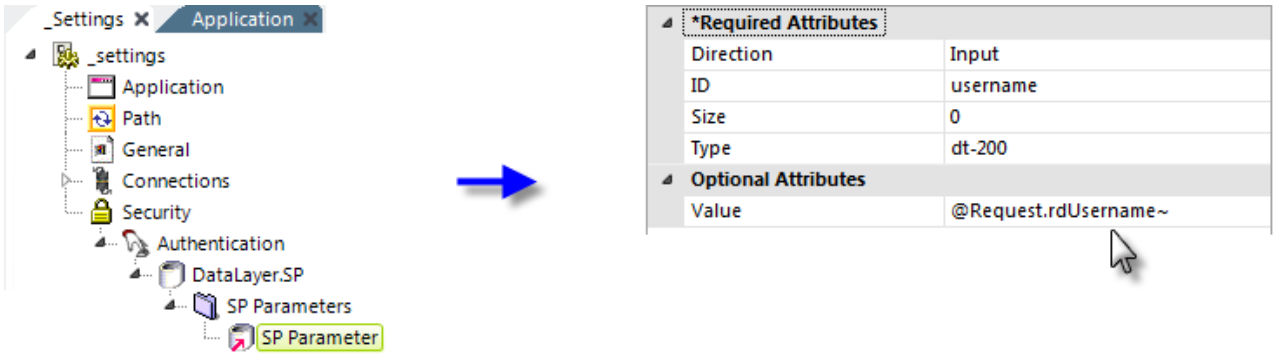


1. In your Logi application's _Settings definition, add a **Security** element, as shown above.
2. Set the **Authentication Source** attribute value to *Standard.*
3. Set the **Security Enabled** attribute to *True*.

The following sections discuss the subsequent steps you'll need to take when using SQL and LDAP authentication methods.

Authenticating Using a SQL Database

Given the potential for a "SQL injection" attack, we highly recommend that you use a *Stored Procedure* to authenticate the submitted credentials.



4. Add an **Authentication** element beneath the Security element, as shown above.
5. Add a **DataLayer.SP** element beneath it and configure it appropriately.
6. Add an **SP Parameters** and an **SP Parameter** element, as shown above, beneath the datalayer. Set the SP Parameter's attributes as shown. The "dt-200" type designation is for VarChar, other types are available in an option list in the attribute. The `@Request.rdUsername~` token contains the username value entered in the standard Login page. Custom login pages should take care to preserve the input control IDs in their forms.
7. Add a second **SP Parameter** element (not shown) and set it similarly for the password, using `@Request.rdPassword~`. Remember: tokens are *case-sensitive*!
8. Your authenticating Stored Procedure should be similar to:

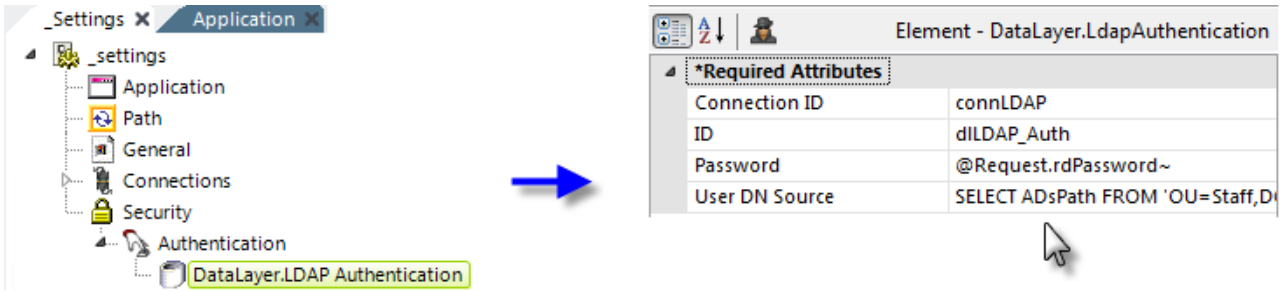
```
@username varchar(20),
@passwordvarchar(16)
SELECT User_Name, User_ID FROM UsersTable
WHERE User_LoginID = LTRIM(RTRIM(@username)) AND User_Password = LTRIM(RTRIM(@password))
```

Parameters are passed to the SP in the *top-to-bottom order* of the SP Parameter elements and do not take the parameter IDs into account. Your SP should "receive" and define the parameters in the same order.

9. Add other Roles and Rights elements as may be necessary (not shown).

If the SP returns *no* data, then the user is *not* authenticated. A successful authentication will return at least one value, which will be automatically assigned internally to the token `@Function.UserName~`. If a second value is returned, it will be assigned to `@Function.UserID~`.

Authenticating Using an LDAP Server



4. Add an **Authentication** element beneath the Security element, as shown above.
5. Add a **DataLayer.LDAP Authentication** element beneath the Authentication element.
6. Set its **Connection ID** attribute value to the ID of a previously configured Connection.LDAP element.
7. Set its **Password** attribute to the appropriate @Request token from the Login page.
8. Set its **User DN Source** attribute to an LDAP query that includes the appropriate @Request token and any other LDAP objects and names appropriate to your LDAP server configuration, such as:

```
(.NET and Standard LDAP Server)
 SELECT ADsPath FROM 'OU=Staff, DC=example, DC=com' WHERE uid='@Request.rdUserame~' AND objectClass-
='InetOrgPerson'
```

```
(.NET and Active Directory Server)
SELECT ADsPath FROM 'DC=myCompanyDomain, DC=local' WHERE
SamAccountName= '@Request.rdUserame~' AND objectClass='organizationalPerson'

(Java and Standard LDAP Server)
 SELECT DN FROM subTreeScope;
WHERE uid='@Request.rdUserame~' AND objectClass= 'InetOrgPerson'

(Java and Active Directory Server)
SELECT CN FROM DC=myCompanyDomain, DC=local WHERE
SamAccountName='@Request.rdUserame~' AND objectClass='organizationalPerson'
```

9. Add other Roles and Rights elements as may be necessary (not shown).

The two @Request tokens mentioned above are passed from the Login page to the application, and any custom logon page substituted for the standard logon page must do the same, using the reserved words "rdUsername" and "rdPassword".

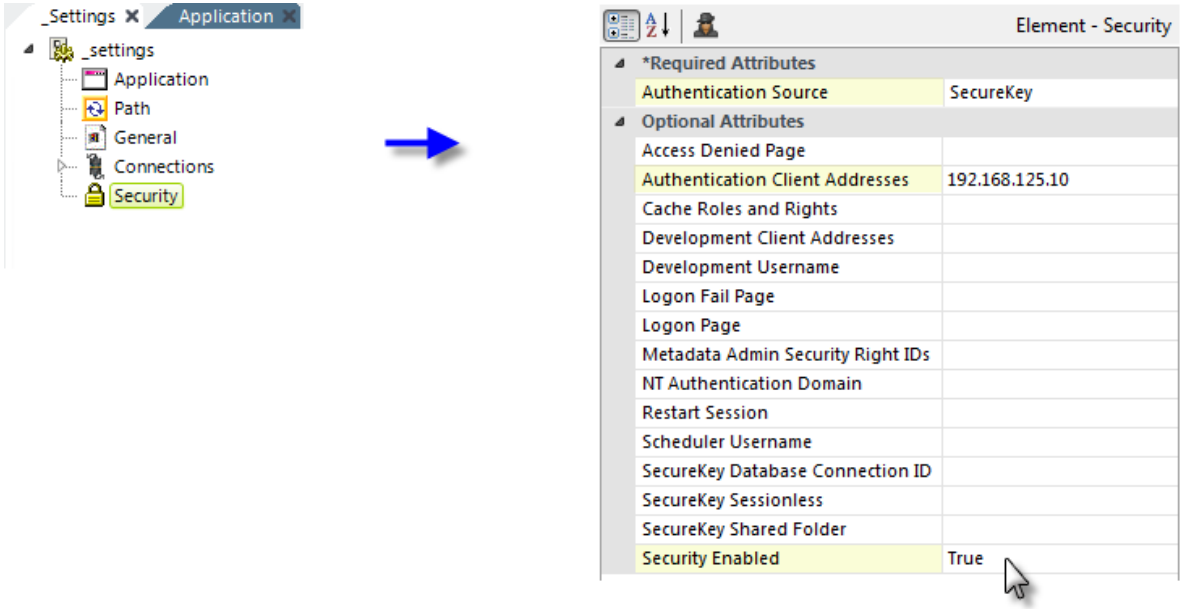
If the query returns *no* data, then the user is *not* authenticated. A successful authentication will return at least one value: the first one will be automatically assigned internally to the token `@Function.UserName~` and, if there's a second one, it will be assigned to `@Function.UserID~`.

Logi Security: SecureKey Authentication

This scenario is used when a Logi report or Logi components are integrated into another application. The other ("Parent") application will present the user with a login page or screen and authenticate the user. It then passes the authenticated credentials, and possibly Role and Rights information, to the Logi application. The key point is that the Parent application authenticates the user.

All that remains to be done is to ensure that a user requesting a Logi report is, in fact, the user who's already been authenticated by the Parent application and Logi SecureKey technology accomplishes this.

To use this type of authentication with IIS, your application's Authentication feature should be configured for *Anonymous Authentication*. Then, to configure your Logi Application for this type of authentication:



1. In your Logi application's _Settings definition, add a **Security** element, as shown above.
2. Set the **Authentication Source** attribute value to *SecureKey*.

3. Set the **Authentication Client Addresses** attribute value to the IP address of the server hosting the parent application. If both the parent and the Logi applications are on the same server, enter 127.0.0.1 here. Multiple addresses can be entered as a comma-separated list, if necessary.

   Computer names may be used, and IP addresses with wildcard masks. To use wildcards, specify an IP address, the space character, and then the wildcard mask. For example, to allow all addresses in the range of 172.16.*.*, specify: `172.16.0.0 0.0.255.255`

   Generic information is available about defining IP wildcard masks..
4. Set the **Security Enabled** attribute to *True*.
5. Add other Roles and Rights elements as may be necessary (not shown).

Modify the parent application to interact with the Logi application: see "Configuring the Parent Application" on page 92 for details.

The tokens `@Function.UserName~`, `@Function.UserRoles~`, and `@Function.UserRights~` can be used to access the data passed from the parent program.

As a debug tool, it's very helpful when developing applications using SecureKey to view the information being sent between the parent application and the Logi application. Test applications, for use in a .NET application only, are available for download from DevNet and help by showing the information in a SecureKey exchange.

Logi SecureKey Authentication

Logi **SecureKey Authentication** technology provides integration for applications requiring secure access to a Logi application or embedded Logi components. It enables a "single sign-on" environment while enhancing security: user authorization is established and requests are made to Logi applications or embedded components using a special security key.

The following topics discuss the use of SecureKey Authentication:

Logi SecureKey is an extension of Logi Security; if you're unfamiliar with Logi Security, first read "Logi Security" on page 25.

Why Use SecureKey Authentication?

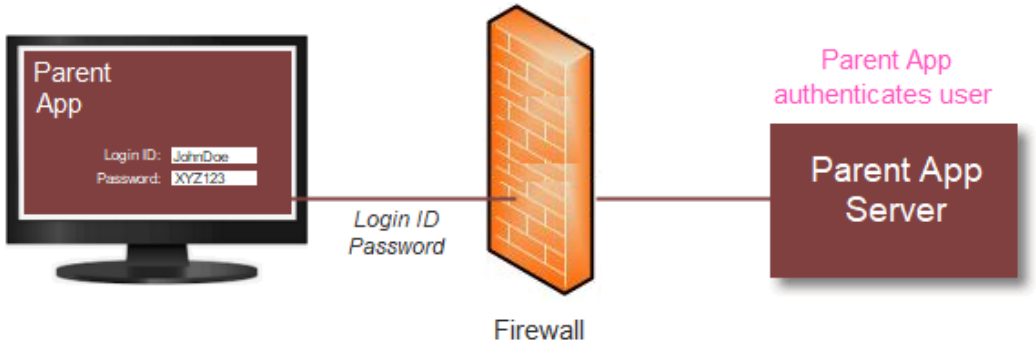
Logi SecureKey Authentication is provided for use in scenarios where a Logi application or Logi components are integrated with or embedded into a non-Logi application (the "Parent" application), which conducts the initial user authentication.

Sometimes called "single sign-on", this arrangement allows users to login once to the Parent application and yet have their security information propagated to other integrated applications, creating a seamless and secure user experience. This, of course, means that users can't be allowed to "go around" the Parent application and directly access the other applications. In the stateless world of web applications, this requires some special mechanisms to ensure security and they are provided for Logi applications through our SecureKey technology.

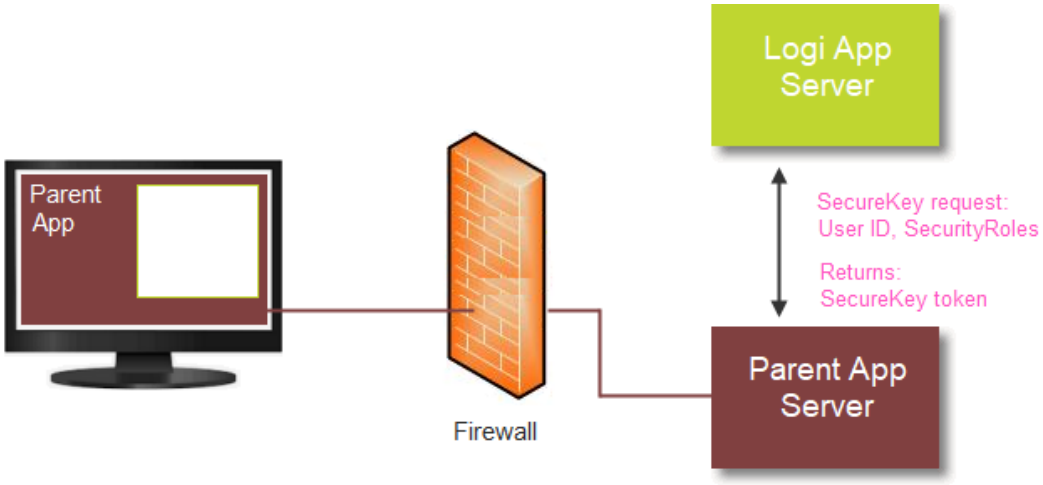
The "authentication" part of SecureKey Authentication refers to the method used by the two applications to verify the validity of requests from users; this is a secondary authentication that occurs separately from the typical standard user login authentication.

How SecureKey Authentication Works

Generally, there are four steps involved in the typical SecureKey Authentication transaction. In the following example scenario, a Logi application is embedded in an iFrame within its Parent application.

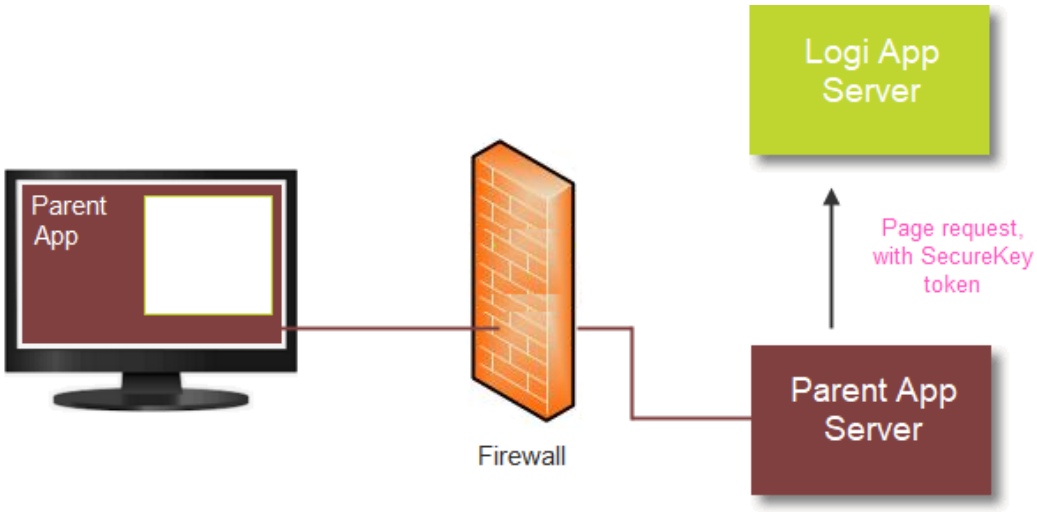


1. The Parent application initially authenticates the user (the "single sign-on"), as shown above. The user successfully logs into the Parent application; the exact method it uses to determine that a user is valid is irrelevant to this process, except that user Roles and/or Rights, if applicable, should be retrieved as part of the authentication.

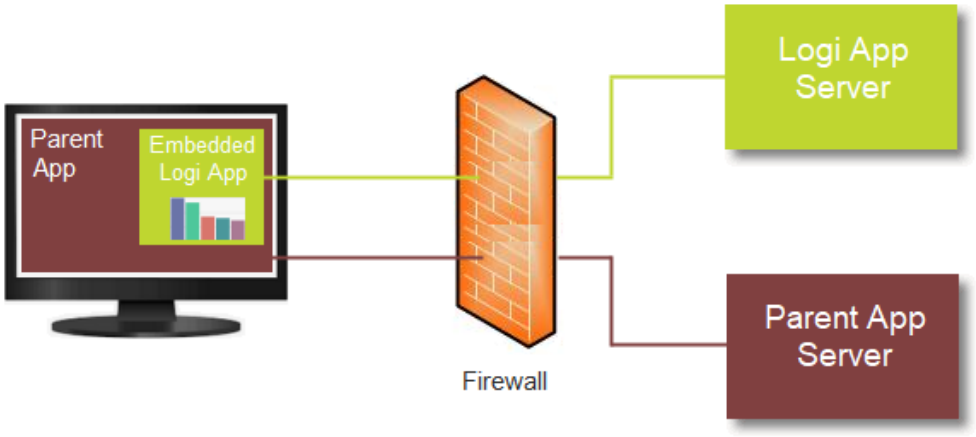


2. After the Parent application authenticates the user, the page that includes the embedded Logi application loads, as shown above. The Parent application sends the Logi application a request for a SecureKey token. The request includes the user ID, any user Roles and/or Rights, and (pre-v12.1.188) the user's machine IP address (the "client browser IP address").

   When it receives the request for a SecureKey token, the Logi application first verifies that the request came from a valid source. This is done by automatically determining the Parent app's *server* IP address and ensuring that it's in a pre-configured list of approved addresses. If so, the Logi application stores the user information from the request and returns a unique SecureKey token to the Parent application.



3. The Parent application then sends the original report request to the Logi application, appending the SecureKey token.



4. Upon receiving the request, the Logi application automatically determines the client browser IP address and uses it, along with the SecureKey token, to authenticate the request. If authenticated, a session is started for the user and any Roles and

Rights (stored in Step #2) associated with the token are used to authorize access to the correct reports, data, and other resources. The requested report is then generated and returned into the iFrame.

When the Logi application session is started for the user, *the SecureKey tokenexpires* and cannot be used again. Any subsequent reports requested from within the context of the Logi application will use the user's session to control access to reports, data, and resources. The user is then communicating directly from his or her browser to the Logi application for reports and the Parent application is no longer involved, as long as the Logi app session persists.

 Developers need to ensure that the Parent application *does not* try to reuse the expired SecureKey token!

If the user's Logi application session ends, and he or she subsequently wants to access reports again, then the process of requesting and using a new SecureKey token will have to be repeated.

Once a SecureKey token is used, it cannot be used again in a different session, and "sharing" tokens across clients isn't possible, as the client browser IP address is referenced within the key when it's created, making it valid *only* for the client that made the initial request.

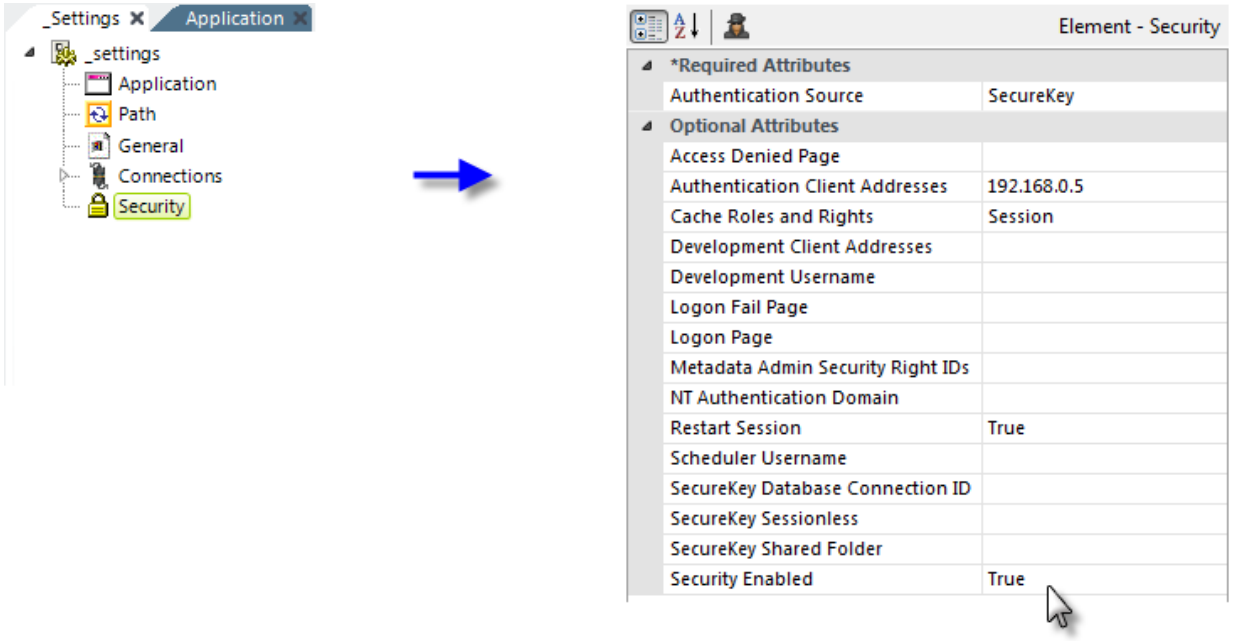
SecureKey Information Storage

Normally, the information associated with a SecureKey token (User Name, Roles, Rights, browser IP address, etc.) is stored as a web server "application scope" variable.

However, if you're using a clustered server configuration, you can specify a physical folder for temporary storage of SecureKey information as XML files. Using a shared network folder for this purpose allows common access to the SecureKey information by the clustered servers. This shared folder is specified in the **SecureKey Shared Folder** attribute of the Security element, in the Logi application's _Settings definition. The SecureKey files created by this option are deleted whenever the automatic "temporary cache file clean-up" occurs (which is configurable - see "Temporary Cache File Management" on page 165).

Configuring SecureKey in a Logi Application

Implementing SecureKey Authentication in a Logi application builds upon existing techniques for implementing **Logi Security**.



In the example shown above, a **Security** element has been added to the application's _Settings definition and configured with typical settings for a SecureKey implementation. The relevant attributes are:

Attribute	Description
Authentication Source	(Required) Select *SecureKey* from the drop-down list of available values.

Attribute	Description
Authentication Client Addresses	Specifies the **IP address** of the web/application *server* making the SecureKey requests (*not* the end-user's IP address), identified as the "Parent App Server" in the first four images above. This value can be determined by "pinging" the application server from the web server hosting your Logi applications. If one computer serves both purposes and the Logi apps are being called using *localhost* in the URL, enter 127.0.0.1 for this value (or ::1 in IPv6). If multiple servers will be making SecureKey requests, multiple IP addresses can be entered here, separated by commas. Computer names may be used, and IP addresses with wildcard masks. To use wildcards, specify an IP address, the space character, then the wildcard mask. For example, to allow all addresses in the range of 172.16.*.*, specify: `172.16.0.0 0.0.255.255` We do not recommend using partial wildcard masks; however, if this set up is absolutely required, you can provide a partial mask as long as the mask value is larger than the IP value. See below for an example. Generic information is available about defining IP address wildcard masks. SecureKey requests from servers with IP addresses *not* in this Authentication Client Addresses list are *rejected*.
Cache Roles and Rights	When Authentication Source is set to *SecureKey* and this value is left blank, the setting defaults to *Session*; SecureKey will not work if it's set to *False*.
Restart Session	When set to *True*, all security-related session variables are cleared with each new login attempt. This prevents session variables that were established during one user's session from being used during a subsequent user's session.

Attribute	Description
SecureKey Database Connection ID	Specifies a database Connection element ID, enabling the temporary storage of SecureKey tokens in a relational database rather than in the file system. The temporary values are stored in a database table named *rdSecureKey*, which is automatically created the first time a SecureKey is used. See this section below for more information.
SecureKey Shared Folder	As discussed earlier, this attribute allows SecureKey to be used with **clustered server configurations** in web farms and web gardens by saving information in commonly-accessible files. The account used by (or impersonated by) the Logi application must have network access rights to read, write, and delete files in this folder. Set the SecureKey Shared Folder value to a network path, such as: `//mySharedServer-/SecureKeyFolder`. Files in the SecureKey shared folder are automatically deleted over time, so do not use this folder to also store other files. If using this attribute for development diagnostics, enter a fully-qualified path on the web server. You can use the `@Function.AppPhysicalPath~` token here as part of the path.
Security Enabled	Must be set to *True* to enable security.

About IP Addresses

Due to the stateless nature of browser interactions, IP addresses play an important role in ensuring security. Developers are occasionally confused about the two IP addresses used with SecureKey authentication, so here's some additional explanation:

Authentication Client Address

This address ensures that the Parent application and the Logi application are allowed to talk to one another. It's the IP address of the *server* hosting the Parent application. It is *not* the IP address of the end-user computer used to request reports through the Parent application.

Computer names may be used, and IP addresses with wildcard masks. To use wildcards, specify an IP address, the space character, then the wildcard mask. For example, to allow all addresses in the range of 172.16.*.*, specify: `172.16.0.0 0.0.255.255`.

This address is *never* passed in a URL when requesting a SecureKey from the Logi application. Instead, the Logi application automatically determines the address of incoming SecureKey requests from the Parent application and, if it matches one of the IP addresses specified in the Security element's Authentication Client Address attribute, then the Logi application will accept and begin processing the SecureKey request.

We do not recommend using partial wildcard masks; however, if this set up is absolutely required, you can provide a partial mask as long as the mask value is larger than the IP value. For example:

`10.158.48.128 0.0.0.218` will work, but

`10.158.48.128 0.0.0.105` will not work.

Client Browser Address

This address is used to verify that each report request to the Logi application is coming from a source that has been authenticated by the Parent application. It's the IP address of the *end-user computer* used to request reports through the Parent application. It

is *not* the IP address of the *server* hosting the Parent application.

This address *is* passed in the URL of every request for a SecureKey. It should *not* be specified in the Security element's Authentication Client Address attribute. The Logi application automatically determines the address of incoming report requests and, if it matches the IP addresses associated with the specified SecureKey, access to the Logi application is allowed.

This query string value is no longer required.

 During development, it's not at all unusual for the Parent application to be hosted by a web server that's on the same machine as the browser used to make test requests. In this case, the Authentication Client Address and the Client Browser Address are very likely going to be the *same*, such as 127.0.0.1 or ::1. This can contribute to the confusion over these two addresses, and perhaps produce a rude surprise when the applications are deployed and the addresses change. We highly recommend testing SecureKey implementations using a separate machine to represent the "end-user machine" in order to avoid this confusion.

Storing SecureKeys in a Database

In clustered environments, the SecureKey Shared Folder is usually used to store SecureKeys so that all computers in the cluster are aware of them. It's also possible to store them instead in a database table, for additional security and to get the file system out of the process. The Security element's **SecureKey Database Connection ID** attribute is used to enable the feature and to provide a connection to the database, which must be on one of these supported database servers:

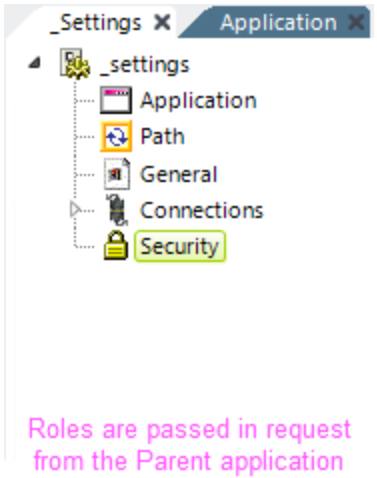
- MS SQL Server
- MySQL
- Oracle
- PostgreSQL

The table, *rdSecureKey*, is automatically created in the database when the feature is enabled and the first SecureKey request is received. SQL scripts for creating the table are also available in `<applicationFolder>\rdTemplate\rdSecureKey`. Temporary SecureKey records, like the temporary files in the file system, are automatically deleted from the table after they expire.

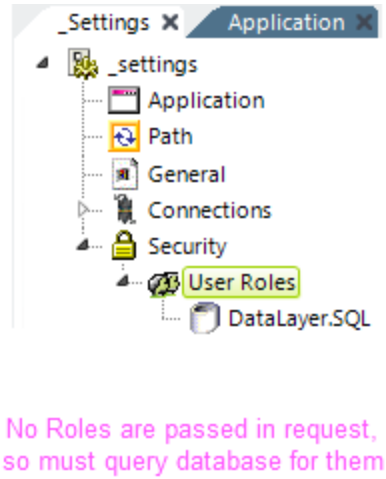
Do not set the **SecureKey Sessionless** attribute to *True* when enabling this feature; an error will result.

Getting User Roles and Rights

Your Logi application may need to use any security Roles or Rights availableas part of the SecureKey information. Various security-related elements are used for this purpose, as follows:



Roles are passed in request from the Parent application



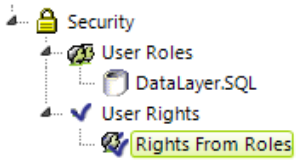
No Roles are passed in request, so must query database for them

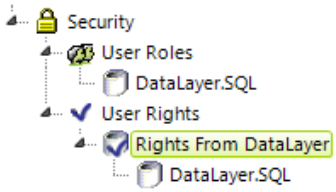
If user Roles information *is* included in the SecureKey request from the Parent application (as described in "Configuring the Parent Application" on page 92), then subsequent report requests will cause that information to be automatically available in the **Security** element, as shown above, left.

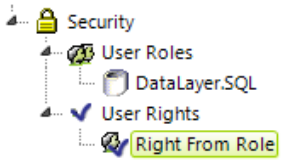
If user Roles information *is not* included in the SecureKey request but is desired, it can still be retrieved from a datasource, using a **User Roles** element and a datalayer, as shown above, right. The data returned into the datalayer should have either a Role value in the first column of each row, or be a single row with one column containing multiple Roles in a comma-delimited list.

If user Rights information *is* included in the SecureKey request from the Parent application (as described in "Configuring the Parent Application" on page 92), then subsequent report requests will cause that information to also be automatically available in the **Security** element.

If user Rights information *is not* included in the SecureKey request but is desired, it can be retrieved in several ways, all of which begin with the addition of the **User Rights** element:







The examples shown above include a User Roles element, but it's not necessary if Roles are supplied to the Security element via the SecureKey request. The separate techniques and elements used to retrieve the Rights are described below:

Element	Technique
Rights From Roles	Builds the list of Rights directly from the Roles values. The Roles and Rights are the same.
Rights From DataLayer	Builds the list of Rights directly from its child datalayer. The data returned should include a Right in the first column of each row, or multiple Rights in a single row ina comma-separated list.
Right From Role	Defines each Right individually, with one Right From Role element for each right. The element's ID is the Right value, which corresponds to Security Right ID attributes in elements in report definitions. Right From Role works with user Roles: if a user has a Role that is returned by this element's child datalayer, then he is allowed

Element	Technique
	the Right specified by the element's ID. The child datalayer should retrieve data with a Role value in the first column of each row, or multiple Roles in a single row ina comma-separated list.

These elements may be used in any combination; the Rights retrieved by each are combined together into a comprehensive rights list.

Configuring the Parent Application

The Parent application is responsible for authenticating the user (the "single sign-on") when they first connect. The authentication process should produce a valid **user ID** and optional lists of Roles (which correspond to Roles established in the Logi application) and/or Rights.

The ClientBrowserAddressquery string valueis no longer required.

When the user first requests a report or web page from the Logi application, the Parent application issues a SecureKey request to the Logi application's authentication service, using this basic format:

```
http://myServer/myLogiApp/rdTemplate/rdGetSecureKey.aspx?Username=bob&ClientBrowserAddress=
192.168.4.75
```

The URL may also optionally include a list of Roles and/or Rights, using this format:

```
http://myServer/myLogiApp/rdTemplate/rdGetSecureKey.aspx?Username=bob&ClientBrowserAddress=
192.168.4.75&Roles=Worker,Manager&Rights=View,Update,Delete
```

The `rdGetSecureKey.aspx` service called in the URL expects or accepts these standard query string values in the URL:

Parameter	Description
Username	(Required) The user ID authenticated during the single sign-on process. In your Logi application this can be referenced using `@Function.Username~`. This token is available for the current user's session and may be referenced in any Logi definition file, including within other child elements of

Parameter	Description
	Security, such as Roles and/or Rights.
ClientBrowserAddress	(Required) The IP address of the requesting user's computer. This IP address is *not* the address of the Parent application server. In the Parent application, this IP address should be determined dynamically; in the example code provided below, for example, the address is determined using Request.UserHostAddress. This query string valueis no longer required. For Java applications, this *must* be in IPv4 format (such as 127.0.0.1) and, if the server supports both IPv4 and IPv6, you may need to force the IPv4 connection by setting this Java Option: `-Djava.net.-preferIPv4Stack=true`. See "Working within NAT or Proxy Environments" on page 98 for additional information relevant to NAT or proxy environments.
Roles	(Optional) A comma-separated list of the security roles for this user, determined during the single sign-on process. This comma-separated list becomes your list of Logi application user Roles. The list can be referenced using `@Function.UserRoles~` throughout the Logi application, and may be used to assign user rights within the security element.
Rights	(Optional) A comma-separated list of the security rights for this user, determined during the single sign-on process. This comma-separated list becomes your list of Logi application user Rights. The list can then referenced as `@Function.UserRights~` throughout the Logi application.

You may also create your own custom query string variables to send other information into the Logi application:

```
http://myServer/myLogiApp/rdTemplate/rdGetSecureKey.aspx?Username=bob&ClientBrowserAddress=
192.168.4.75&Roles=Worker,Manager&myCustomParam1=10&myCustomParam2=120
```

These extra values will be assigned to session variables in the Logi application and can be accessed using @Session tokens. For example, `@Session.myCustomParam1~` would equal `10` and `@Session.myCustomParam2~` would equal `120`. Remember that theses tokens are case-sensitive and you will need to match the spelling in the query string *exactly* when using them.

The ClientBrowserAddress query string valueis no longer required.

Using POST vs GET

For additional security, or if you have a lot of custom query string variables or really long values, you may elect to POST the URL and its query parameters. The Logi application will examine the HTTP request for them automatically, regardless of whether they were sent using HTTP POST or GET.

Using the SecureKey

The Logi application's authentication service returns a SecureKey to the Parent application, which then *redirects* the user's request to the Logi application, passing the SecureKey as an additional query string parameter. The URL for this redirect will look something like this, using the two keywords *rdReport* and *rdSecureKey*:

```
http://myServer/myLogiApp/rdPage.aspx?rdReport=myDesiredReport&rdSecureKey=1a34c56f7b89
```

Sample Code

The following Visual Basic.NET code provides an example of the two things you'll need to do in a Parent application: request a SecureKey and redirect the user's report request using the SecureKey.

Your Parent application may have multiple links to the Logi application, so the following code is an example of a function that can be called from the click event handlers for any of those links. It assumes the user name, user Roles, and user Rights have been placed into session variables as part of the login authentication process and that the Logi application base URL has been specified in the `Web.config` file.

```
Public Function getSecureKey() As String

' define parts of the SecureKey request URL
' Logi application base URL is set in Web.config file
Dim strLogiAppBaseURL As String = ConfigurationManager.AppSettings("LogiAppBaseURL")
Dim strGetKeyURL As String = "/rdTemplate/rdGetSecureKey.aspx?Username=" & Session("UserName") & "&Roles="_
 & Session("UserRoles") & "&Rights=" & Session("UserRights")
Dim strClientAddr As String = "&ClientBrowserAddress=" & Request.UserHostAddress

' define web request and response receiver
Dim webRequest As Net.HttpWebRequest
Dim webResponse As Net.WebResponse|
webRequest = Net.HttpWebRequest.Create(strLogiAppBaseURL & strGetKeyURL & strClientAddr)

' send the SecureKey request to the Logi app
Try
```

```
webResponse = webRequest.GetResponse()
Catch ex As Exception
' if server has Basic or NTLM authentication, try to set creds from the current process
If ex.Message.IndexOf("401") <> -1 Then
webRequest.Credentials = Net.CredentialCache.DefaultCredentials
webResponse = webRequest.GetResponse()
Else
Throw ex
End If
End Try

' receive the response and return it as function result
Dim sr As New IO.StreamReader(webResponse.GetResponseStream())
getSecureKey = sr.ReadToEnd()

End Function
```

The event handler code for the links to the Logi application might look like this:

```
Protected Sub lbtnReports_Click(ByVal sender As Object, ByVal e As EventArgs) Handles lbtnReports.Click

' identify the Logi report definition to be displayed by this menu item or link
Dim strLogiDefName As String = "Default"

' get the Logi application base URL, which is set in the Web.config file
```

```
Dim strLogiAppBaseURL As String = ConfigurationManager.AppSettings("LogiAppBaseURL")

' get a SecureKey for this request
Dim strSecureKey As String = getSecureKey()

'redirect to Logi app
Response.Redirect(strLogiAppBaseURL & "/rdPage.aspx?rdReport=" & strLogiDefName _
 & "&rdSecureKey=" & strSecureKey)

End Sub
```

Additional coding examples in other languages are available in "SecureKey Coding Examples" on page 101.

Debugging SecureKey Requests

It can be very helpful, when developing applications, to view the information being sent between the Parent application and the Logi application during the SecureKey request.

To do this, temporarily provide a value for the Security element's **SecureKey Shared Folder**. This will cause SecureKeys to appear there as XML files, so you can see when they're created. You can open and examine them in order to determine what's being passed in the SecureKey requests.

We also offer a SecureKey test program (.NET environment only) which is available for download from DevNet; it displays the SecureKey information in a test exchange: https://clm.logianalytics.com/downloads/info/TestSecureKey.zip

Working within NAT or Proxy Environments

If a company has set up a firewall with NAT'd addresses or another form of network proxy that sends all traffic to the Logi application through a single IP address or a range of IP addresses, then client IP address checking in SecureKey needs to be suppressed because individual user locations will not be unique. This can be achieved by modifying the Parent application's request to the Logi application so that it passes a Client Browser Address value of 0.0.0.0. For example:

```
http://myServer/myLogiApp/rdTemplate/rdGetSecureKey.aspx?Username=bob&Roles=Worker,Manager &Cli-
entBrowserAddress=0.0.0.0
```

When using this approach, security can be further enhanced by limiting requests to those from certain parts of the network by using an IP address mask in the web server's security configuration. For IIS, this is accomplished in the IIS Manager's Directory Security tab; for Tomcat this is accomplished by adjusting some of its XML configuration files. The ClientBrowserAddress query string value is no longer required.

Managing SecureKey Sessions

This topic introduces ways to manage your SecureKey sessions.

Application Server Time-out Configuration

Automatic **sessiontime-out** after a period of inactivity is managed in the web server and, for example, the IIS default setting is 20 minutes. You may want to adjust this to a shorter interval in order to increase session security, which is done in a variety of ways. For example, for IIS, this is set in IIS Manager tool or in the `web.config` or `machine.config` files; for Tomcat, it's set in the `web.xml` file. While this, by itself, is not completely secure (it still presents a window of opportunity until the time-out occurs), it's a good practice that helps minimize security exposure.

Session Exit Elements

The **Action.Exit** and **Response.Exit** elements immediately end the current session and redirect the browser to a URL. In Logi Info, you can initiate this from your parent application by browsing directly to a process task in the Logi app, which runs Response.Exit. Otherwise, both actions are usually initiated by a user action, such as clicking a "Logout" button or link (however, user behavior in this regard may not always be reliable).

Restart Session Attribute

The Security element's **Restart Session** attribute causes previous session information to be cleared with each new SecureKey authentication attempt. Turning this feature on is the recommended method of ensuring that users do not "piggyback" onto a previous user's session.

This requires, however, that your parent application be designed correctly: the authentication routine in the parent app *always* needs to include a Logi SecureKey authentication request. It should not, for example, include code that tests to see if a Logi session exists and then skips issuing a new SecureKey request if it does. Ensuring that there's a new SecureKey request with each report request will guarantee that other sessions that are not valid (such as those created by using browser bookmarks or shared links) will be rejected and redirected back to the parent app for authentication. It will also ensure that different users on the same machine/browser will also have to authenticate properly (providing that the original user logs out of the parent application session).

SecureKey Coding Examples

This topic helps developers who are writing applications that will call a Logi application using Logi SecureKey Authentication.

Language examples include:

- C# (.NET)
- Visual Basic (.NET)
- Java
- Node
- Perl
- PHP
- Python
- Ruby

About Logi SecureKey Authentication

Logi **SecureKey Authentication** is a technology we offer for use in "single sign-on" scenarios, where a non-Logi application (the "Parent" application), which conducts the initial user authentication, is integrated with a Logi application. SecureKey Authentication provides a method of securely accessing the integrated Logi app, and is discussed in detail in "Logi SecureKey Authentication" on page 77. This topic provides examples of the Parent application code required to establish the flow of secure exchanges with a Logi application using SecureKey.

The examples, in several languages, are very generic and are only intended to provide the basic, necessary code. They're provided with the assumption that developers who undertake using them and extending their functionality have in-depth skills with the language selected. Other languages can also be used and success with them will, again, depend on the developer's skills. The ClientBrowserAddress query string value is no longer required.

C# (.Net)

The following code example is written in C#, for use with the .NET framework:

```
/* set up the SecureKey call */
// define parts of the SecureKey request URL
string strLogiAppBaseURL = "http://<yourServer>/<LogiApp>";
string UserName = "Bob";
string Roles = "Admin";
string Rights = "Manager";
string ClientBrowserAddress = Request.ServerVariables["REMOTE_ADDR"];

// build complete URL to Logi app
```

```
string sGetKeyURl = "/rdTemplate/rdGetSecureKey.aspx?Username=" + UserName + "&Rights=" + Rights + "&Roles=" + Roles + "&ClientBrowserAddress=" + ClientBrowserAddress;

string sFinalURL = strLogiAppBaseURL + sGetKeyURl;



// create and send SecureKey request

string sKey = "";

HttpWebRequest webRequest = (HttpWebRequest)HttpWebRequest.Create(sFinalURL);

HttpWebResponse webResponse;



// attempt to make SecureKey call, read return into sKey

try

{

webResponse = (HttpWebResponse)webRequest.GetResponse();
```

```
Stream myStream;
myStream = webResponse.GetResponseStream();
StreamReader sr = new StreamReader(myStream);
sKey = sr.ReadToEnd();
}
catch (Exception ex) // if not bad URL, send error message
{
if (ex.Message.IndexOf("401") != -1)
{
Response.Write("Not Found");
}
else
{
```

```
35.  Response.Write(ex.Message.ToString());
36.  }
37.  }
```

Visual Basic (.NET)

The following code example is written in Visual Basic,for use with the .NET framework:

```
' set up the SecureKey call
' define parts of the SecureKey request URL
Dim strLogiAppBaseURL As String = "http://<yourServer>/<LogiApp>"
Dim strUserName = "Bob"
Dim strUserRoles = "Admin"
Dim strUserRights = "Manager"
Dim strClientBrowserAddr As String = Request.UserHostAddress

' build complete URL to Logi app
```

```
10. Dim strGetKeyURL As String = "/rdTemplate/rdGetSecureKey.aspx?Username=" & strUserName & "&Roles=" & strUser-
    Roles & "&Rights=" & strUserRights & "&ClientBrowserAddress=" & strClientBrowserAddr
11. Dim strFinalURL As String = strLogiAppBaseURL & strGetKeyURL
12.
13. ' define web request and response receiver
14. Dim webRequest As Net.HttpWebRequest
15. Dim webResponse As Net.WebResponse
16. webRequest = Net.HttpWebRequest.Create(strFinalURL)
17.
18. ' attempt to make SecureKey call
19. Try
20. webResponse = webRequest.GetResponse()
21. Catch ex As Exception
22. ' if server has Basic or NTLM authentication, try to set credentials from the current process
```

```
23.  If ex.Message.IndexOf("401") <> -1 Then
24.  webRequest.Credentials = Net.CredentialCache.DefaultCredentials
25.  webResponse = webRequest.GetResponse()
26.  Else
27.  Throw ex
28.  End If
29.  End Try
30.
31.  ' receive the response and return it as function result
32.  Dim sr As New IO.StreamReader(webResponse.GetResponseStream())
33.  getSecureKey = sr.ReadToEnd()
```

Java

The following code example is written in Java,for use with a JSP file:

```
<%@ page language="java" import="java.util.*,java.net.*,javax.servlet.*,java.io.*" %>

<%
// set up the SecureKey call
// define parts of the SecureKey request URL
String LogiAppURL;
String Username;
String Roles;
LogiAppURL = "http://<yourServer>/<LogiApp>";
Username = "Bob";
```

```
11.  Roles = "1";
12.
13.  // build complete URL to Logi app
14.  String getKeyURL = "/rdTemplate/rdGetSecureKey.aspx?Username=" + URLEncoder.encode(Username) + "&Roles=" +
     URLEncoder.encode(Roles) + "&ClientBrowserAddress=" + URLEncoder.encode(request.getRemoteAddr());
15.  String finalURL = LogiAppURL + getKeyURL;
16.
17.  // back-end handshake, passing username, roles and rights to Logi app
18.  URL url = new URL(response.encodeRedirectURL(finalURL));
19.
20.  // attempt to make SecureKey call
21.  try {
22.  // open the connection
```

```
23.  URLConnection conn = url.openConnection();
24.  conn.setDoOutput(true);
25.
26.  // debugging aid: use StreamWriter to write POST data
27.  // OutputStreamWriter wr = new OutputStreamWriter(conn.getOutputStream());
28.  // wr.write(data);
29.  // wr.flush();
30.  // wr.close();
31.
32.  // get the response
33.  BufferedReader rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
34.  StringBuffer sb = new StringBuffer();
35.  String line;
```

```
while ((line = rd.readLine()) != null) {
sb.append(line);
}
rd.close();


// set the url with the retrieved SecureKey parameter
LogiAppURL = LogiAppURL + "/rdPage.aspx?rdSecureKey=" + sb.toString();


// redirect to the application with SecureKey parameter
response.sendRedirect(LogiAppURL);
}
catch (Exception e) {
throw new Exception("Failed to get SecureKey, request URL: [" + url + "], POST data: [" + data +"]", e);
```

```
49. }
50. %>
```

Node

The following code example is written in Node.js:

```
var express = require("express");
var http = require("http");
var app = express();


var LogiAppURL = "http://<yourServer>/<LogiApp>";
var UserName = "Bob";
var Roles = "Admin";
var Rights = "Manager";
var MessageBack = "";

```

```
/* create an app to listen on the port # specified in the app arg */

app.get("*", function(req, resM) {



/* send the SecureKey request */

http.get(LogiAppURL + "/rdTemplate/rdGetSecureKey.aspx?"Username=" + UserName + "&Roles=" + Roles + "&Rights="
+ Rights + "&ClientBrowserAddress=" + resM.connection.remoteAddress, function(res) {



/* when the key is returned */

res.on("data", function(chunk) {

MessageBack = chunk;



/* send a response to the browser (using our Embedded Report API in this example) */

resM.send("<p style=\"font-size:20px;text-align:center;\" > This is a node page </p> <div id=\"divNd\" data-autoS-
izing=\"all\" data-applicationUrl=\"' + LogiAppURL + '\" data-report=\"Default\" data-linkParams=\"{'rdSecurekey' : '" +
```

```
MessageBack + "'}\" > </div> <script src=\"' + LogiAppURL + '/rdTemplate/rdEmbedApi/rdEmbed.js\" type-
e=\"text/Javascript\"></script>"
);

});


/* server-side error logging */

}).on('error', function(e) {

console.log("Got error: " + e.message);

});

});


console.log("Web application opened.");

app.listen(1337);
```

Perl

The following code example is written in the Perl language:

```
#!/usr/local/bin/perl


# Requires LWP to be installed
use LWP::Simple;


# define parts of the SecureKey request URL
my $logiAppUrl = 'http://<yourServer>/<LogiApp>';
my $user = 'Bob';
my $role = 'Admin';
my $rights = 'Manager';
```

```
my $clientAddress = print $ENV{REMOTE_ADDR};



# build complete URL to Logi app

my $url = $logiAppUrl . '/rdTemplate/rdGetSecureKey.aspx?Username=' . $user . '&Role=' . $role . '&Rights=' . $rights . '&ClientBrowserAddress=' . $clientAddress;



# make the SecureKey call and get the response

my $response = get $url;



# The response should be used as the rdSecureKey parameter value in HTML or Javascript in a Div or iFrame

print $response;
```

PHP

The following code example is written in the PHP language:

```
<?php
// Application has Authenticated the user and has access to Roles and Rights
// Generally the Application would now place the username, roles and IP address into variables
// The client's IP address can be found with: $_SERVER['REMOTE_ADDR'];


// Getting SecureKey from Logi Info Application (again the Username, roles and IP address can be variables)
$logiURL = "http://path/to/app";
$username = "bob";
$roles = "admin";
$rights = "king";
```

```
11.


12.  // Creating URI for SecureKey Call

13.  $skeyCall = "rdTemplate/rdGetSecureKey.aspx?Username=" . $username . "&Roles=" . $roles . "&ClientBrowserAddress="
     . $_SERVER['REMOTE_ADDR'];

14.  $ch = curl_init($logiURL.$skeyCall);
15.


16.  curl_setopt($ch, CURLOPT_RETURNTRANSFER, true); curl_setopt($ch, CURLOPT_BINARYTRANSFER, true); $secureKey=
     curl_exec($ch); curl_close($ch);
17.


18.  // Alternate way of getting info. Sometimes blocked due to security issues

19.  // $secureKey= file_get_contents('http://myServer-
     /myLo-
     giApp/rdTemplate/rdGetSecureKey.aspx?Username=bob&Roles=Worker,Manager&ClientBrowserAddress=192.168.4.75')
20.
```

```
// Output the JS Embedded API control

echo "<div data-autoSizing='all' data-report='default' data-applicationURL='$logiURL'></div>";

echo "<script src=' + LogiAppURL + '/rdTemplate/rdEmbedApi/rdEmbed.js' type='text/Javascript'></script>";


// Output the iframe control

// echo "<div>";

// echo "<iframe id=\"iframe1\"src=\"http://127.0.0.1/SecureKey/rdPage.aspx?rdSecurekey=" . $secureKey . "\" frame-border=\"0\" onLoad=\"autoResize('iframe1');\" scrolling=\"no\" />";

// echo "</div>";

?>
```

Python

The following code example is written in the Python language:

```
# import "requests" library, for http requests
import requests


# define parts of the SecureKey request URL
class classLogiSecureKeyURL:
  def __init__(object, url, user, roles, rights, ipaddress):
    object.cLogiAppUrl = url
    object.cUser = "?Username=" + user
    object.cRoles = "&Roles=" + roles
    object.cRights = "&Rights=" + rights
```

```
    object.cIPAddress = "&ClientBrowserAddress=" + ipaddress

    object.cSecureKeyUrl = url + "/rdTemplate/rdGetSecureKey.aspx" + "?Username=" + user + "&Roles=" + roles + "&Rights=" + rights + "&ClientBrowserAddress=" + ipaddress


# make the SecureKey call and get the response
 def getSecureKey(classLogiSecureKeyURL):
    try:
        r = requests.get(classLogiSecureKeyURL.cSecureKeyUrl)
    except requests.exceptions.RequestException as e:
        print(e)
        sys.exit(1)
    # print Secure Key request URL
    print("Secure Key request URL:\n")
```

```
    print(classLogiSecureKeyURL.cSecureKeyUrl)
    print("\n\n")
    # print Secure Key
    sKey = r.text
    print("Secure Key:\n")
    print(sKey)
    return r


# The response should be used as the rdSecureKey parameter value in HTML or Javascript in a Div or iFrame
sKey = classLogiSecureKeyURL("http://localhost/MyNetApplicationName", "admin", "Manager", "Manager", "0.0.0.0")
sKey.getSecureKey()
```

Ruby

The following code example is written in the Ruby language:

```
require 'net/http'


# define parts of the SecureKey request URL
LogiAppURL = "http://<yourServer>/<LogiApp>"
username = "Bob"
roles = "Admin"
rights = "Manager"
clientaddr = Socket.unpack_sockaddr_in(socket.getpeername) >> ip  # ruby v1.9


# build complete URL to Logi app
```

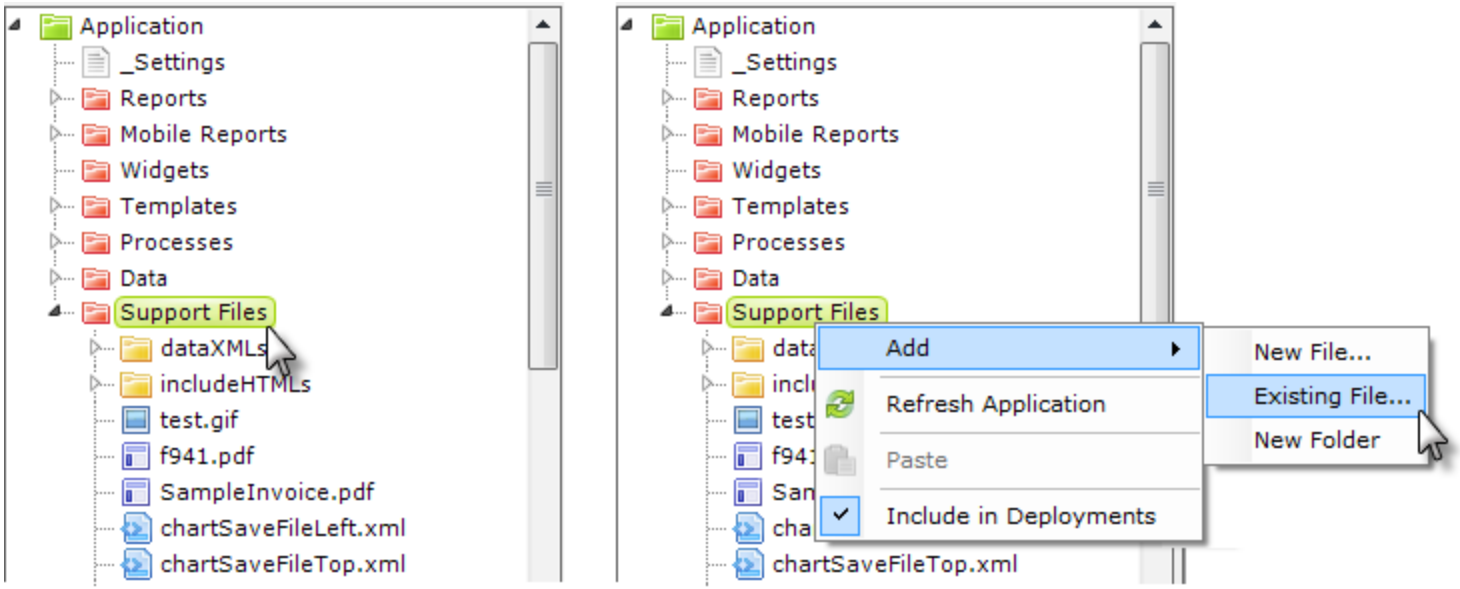
```
11. getKeyURL = "/rdTemplate/rdGetSecureKey.aspx?Username=" << username << "&Rights=" << rights+ "&Roles=" <<
    roles << "&ClientBrowserAddress=" << clientaddr
    finalURL = LogiAppURL << getKeyURL
12.



13. # make the SecureKey call and get the response

14. result = Net::HTTP.get_response(URI.parse(finalURL))

15. sKey = result.body

16. print sKey
```

Support Files Management

In Logi applications, "Support Files" are defined as files that are part of an application that are not **Report**, **Process**, or other definition files. This topic discusses support files and their management within an application and provides important information about how files are handled in Studio. Support files generally include:

- Images (.gif, .jpg, .jpeg, .png, .bmp, .wmf, .xbm, .art)
- Style Sheets (.css)
- Script files (.js)
- Data files (.csv, .json, .mdb, .xls, xlsx, .xml)
- HTML files (.htm, .html)

Logi Studio is designed to make managing these files as easy as possible.



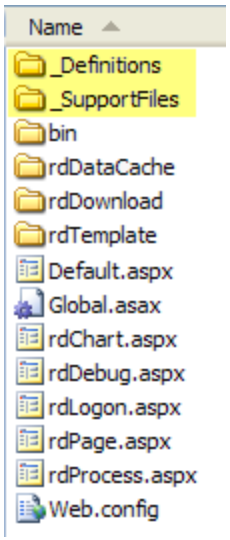
In Logi Studio, support files are managed in the **Application** Panel. The panel displays standard organizational folders for each category of file in an application, including Support Files. The default manner in which support files are displayed is shown above left, with all support files shown together.

In addition to the file types mentioned earlier, you're free to store other files of other types there, such as .pdf or .docx, if it's convenient for you.

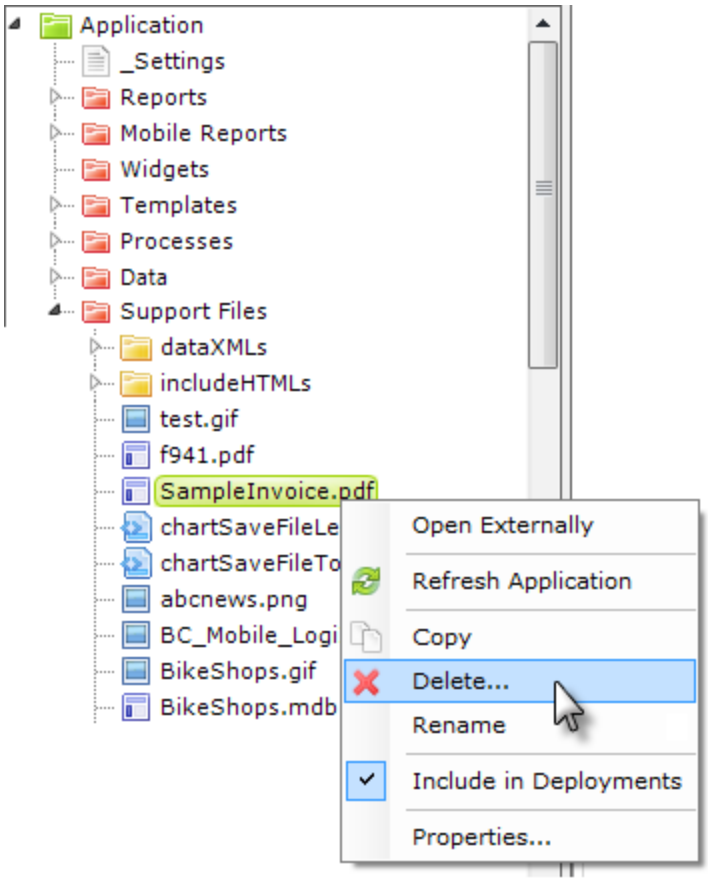
Right-clicking the Support Files folder title in Studio produces a pop-up menu, as shown above, right. Adding an existing file *copies* the existing file to your project folder; the original file is left untouched.

The Application Panel displays the files that are included in the application; there are no Registry entries, tables, or other data concerning the files maintained by Studio as part of an application. This lets you move, rename, or delete files *outside* of Studio if you want to, without causing havoc.

Adding a large number of support files from within Studio could be time-consuming. To save time in this situation, files can be copied outside of Studio, using file system tools, directly into the appropriate folder. The files will automatically appear in Studio the next time the application is refreshed (using the pop-up menu option), or the application is closed and re-opened.

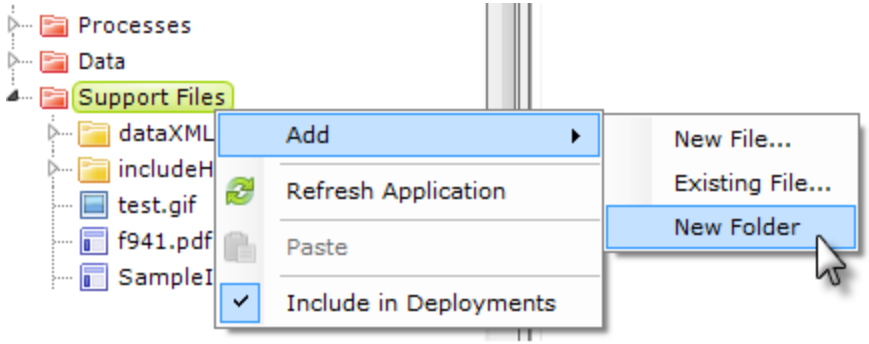


The image above shows a typical collection of files and folders for a basic Logi application, as they exist in the file system. The top two folders, highlighted in yellow, correspond to the **Reports** and **Support Files** folders shown in Studio's Application panel. Other empty folders shown in Studio, such as Templates, will be physically created in the file system automatically when they're given some contents in Studio.



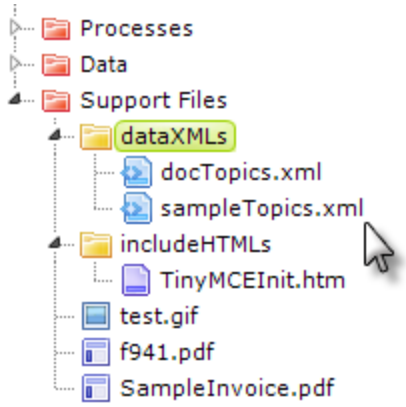
Studio is equipped with special editors in the **Workspace** panel for the text-based support file types (.css, .js, .json, .xml, .htm, and .html). Double-clicking these files in the Application panel will open them in the Workspace panel for editing. If you double-click a non-text file type, such as .xlsx, it will open in the external application (in this case Excel) associated with it in the file system.

You can also manage any file (copy, delete, rename, etc.) from within Studio by right-clicking it, as shown above.



If you wish to further organize your support files, you can do so by creating a new "folder", as shown above. This is a logical folder and does not appear in the file system as a separate folder. Here's how this works: when you create a new logical folder, you can give it a name, such as "dataXMLs".

Files that you add to that folder from within Studio will appear in the file system with the folder name pre-pended to their name, with a period separator. Or, if you're working directly in the file system, to get files to appear in that folder, just rename them to begin with the folder name and a period.



So, for example, renaming `sampleTopics.xml` to `dataXMLs.sampleTopics.xml` in the file system will cause it to appear in Studio "inside" the new dataXMLs folder you created.

 You're *not* required to manage support files from within Studio, but by doing so you ensure that all necessary files are included in the project folder with the proper access permissions, simplifying deployment. In addition, specifying the location of files within your report definitions is simplified as the standard support file folders are "known" to Logi applications.

Session Variables

Session Variables are values that are global in scope and persistent for the life of a user session. They're very useful in Logi applications for retaining and passing data. This topic discusses the elements and techniques associated with Session Variables in a Logi application.

The following sections discuss working with Session Variables:

- Setting Session Variables
- Using @Session Tokens
- Java Session Variable Copying
- Viewing Session Variables in Debugger

About Session Variables

Session Variables provide a useful way to store values that can be used anywhere in a Logi application and provide the best approximation of a real programming *variable* you'll find in stateless web applications. Logi application Session Variables can be set in both Report and Process definitions using the **Set Session Variables** and **Procedure.Set Session Vars** elements, respectively. They can also be set by other, non-Logi applications and used in an integrated Logi application (see the Java Session Copying section for special considerations that apply to Java applications).

Session Variables, of course, only exist as long as the user session lasts. The Session Timeoutelement can be used to manage the user session lifetime. When using "Logi SecureKey Authentication" on page 77, the **Security** element in _Settings includes a **Restart Session** attribute that can be set to clear out all Session Variables right before an attempted login. This ensures that Session Variables established during a previous user's session are not re-used for the next user's session.

Restrictions

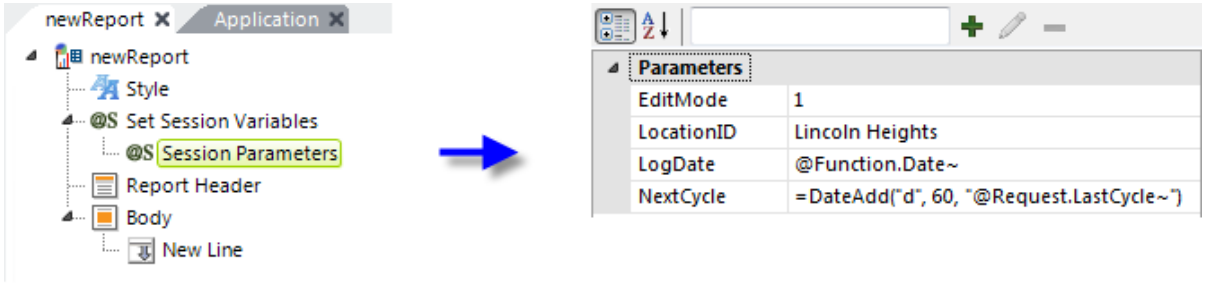
Session Variables are of the *variant* data type, meaning they can be anything - strings, numbers, arrays, etc. - of any size, and that opens the door to potential abuse. It's possible to put huge amounts of data into Session Variables but it's poor practice to do so, because that will negatively affect server performance. Our recommendation is that you use Session Variables for *small amounts of data only*.

The **Repeat Elements** element is one of the very first things processed when a page goes through the Logi Engine. Some values, including @Session tokens set with the two Set Session elements mentioned earlier, aren't processed until *later* so @Session tokens should *not* be used with Repeat Elements.

Setting Session Variables

Setting Session Variables in a Report definition is very easy:

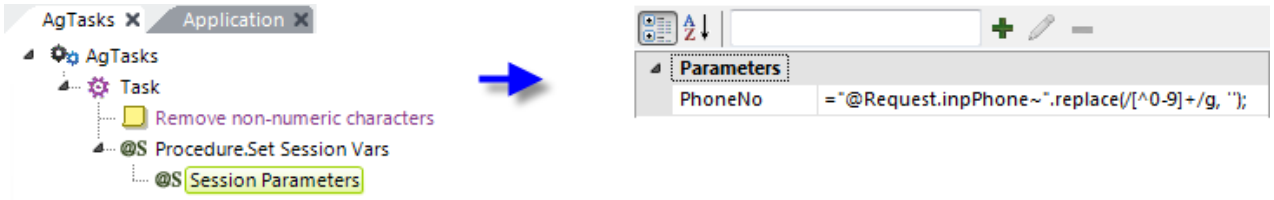
First, the **Set Session Variables** element is added to the definition, as shown above. Its optional attributes are also shown and can be very useful. The **Condition** attribute lets you specify a formula which has to evaluate to *True* for the Session Variables to be set. The **Set First Time Only** attribute, if set to *True*, causes the Session Variables to be set the first time the definition is processed but does not reset them if the report is reloaded during the current user session. The default value is *False*.



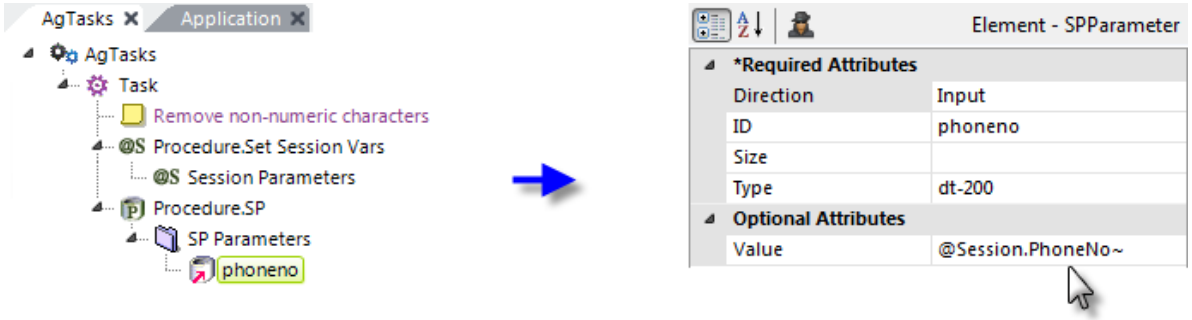
Next, a **Session Parameters** child element is added. This element lets you enter a list of Session Variable names and their values. As shown above, the values can be numbers, strings, tokens, or even formulae. A formula can be particularly useful here, especially if you want to make the Session Variable value dynamic in order to, for example, increment a counter or a date each time the report is reloaded. This is one of the few opportunities you have to make a web application variable behave like a normal programming variable.

Setting Session Variables in a Process Task

In the next example, we'll examine a similar operation in a Process definition (only available in Logi Info).



In a task shown above, a **Procedure.Set Session Vars** element has been added, with a child **Session Parameters** element. A Session variable, "PhoneNo", has been created, with a value that uses a formula to remove any non-numeric characters from a Request variable passed to the task.



Later in the task, we can see how the @Session token is used to provide the "cleaned up" phone number value as a parameter for a SQL Stored Procedure. As we've seen, Session Variables can be used in Report and Process definitions to produce very dynamic, global values.

Setting Session Variables in Global.asax

In a .NET Logi application, you can also set Session Variables in the application's `Global.asax` file by adding code like this:

```
<% Session("webpath") =
 Request.ServerVariables("URL") %>
```

This code takes advantage of ASP.NET objects to assign the URL of the application to the "webpath" Session variable. In the application, the value can be accessed using the `@Session.webpath~` token.

Using @Session Variables

@Session tokens can be employed in a lot of places in your definitions. Here are some more usage examples:

Navigation Persistence

Because Session Variables are persistent across pages, they can be particularly useful in preserving values just before a Process task call. For example, suppose you have a task for user login that users can call from any page in your application. After the login task completes, you want to return the user to the page he was viewing. Here's one technique for doing this in a parameterized manner:

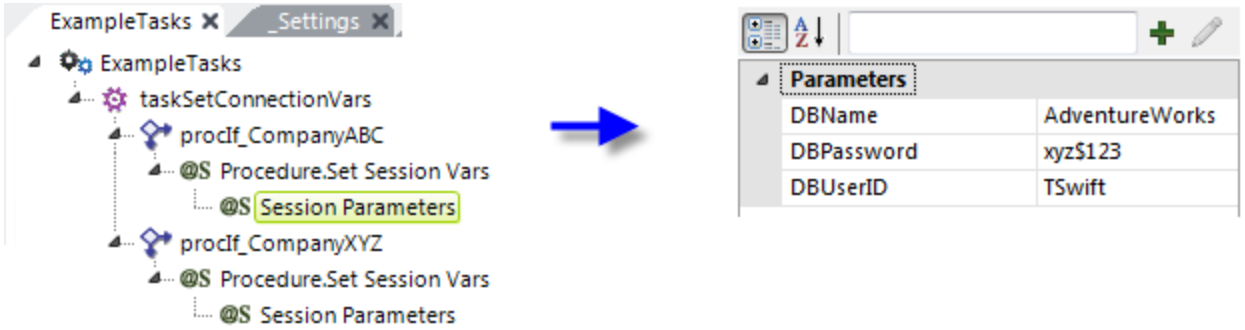
1. At the top of each report definition, use elements to create a Session Variable ("Caller") with a value of `@Request.rdReport~`. This is the definition name of the report.

2. Each report definition, of course, has a link to the Login task.

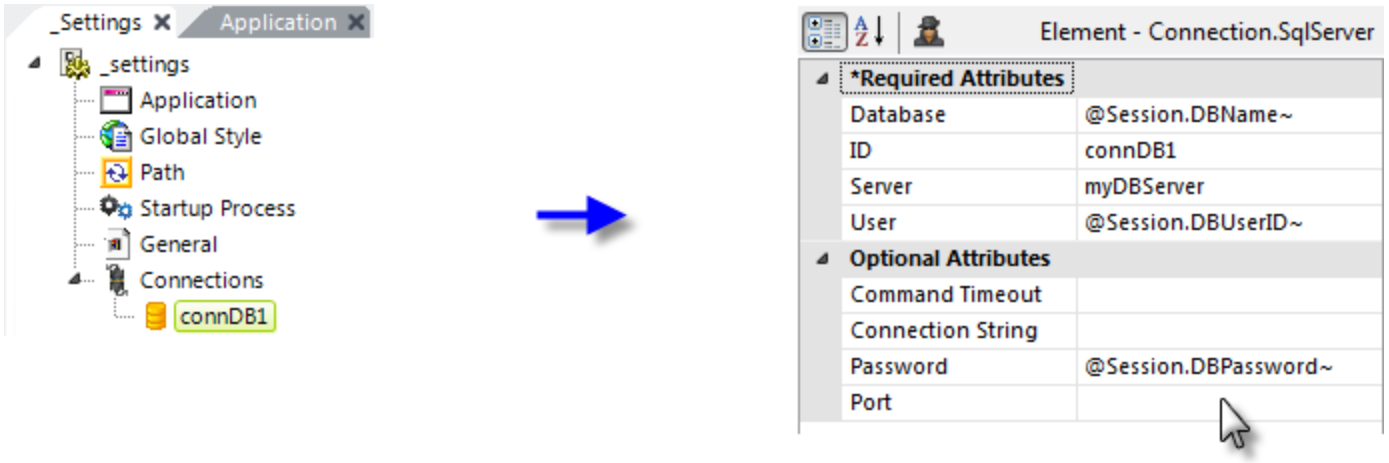
3. At the end of the Login task, use Response.Report and Target.Report elements. In the Target.Report element's Report Definition File attribute, enter the `@Session.Caller~`token. This ensures the user will go back to the page he was viewing before calling the task.

Dynamic Connections

@Session tokens can be used in the attributes of **Connection** elements, allowing you to create dynamic connections. Typically, this is done by using a **Startup Process** element in _Settings to run a Process task.



As shown above, the task uses **Procedure.If** and **Procedure.Set Session Vars** elements to set appropriate Session Variables based on variable criteria.



When the task completes, the Connection elements in _Settings will be processed, using @Session tokens, as shown above.

Dynamic JavaScript Code

Like all tokens, @Session tokens can be used in JavaScript code to make it dynamic:

```
if ( '@Session.UserID~' == '' ) alert('Your session has expired.');
if ( '@Session.UserID~' != '' )
 SubmitForm('rdProcess.aspx?rdProcess=someTaskID&UserID=@Session.UserID~',
    '_self','true','',null);
```

The code above embeds @Session tokens to determine if a user ID is present and calls a Process task if it is.

Java Session Variable Copying

Java-based Logi applications and non-Logi Java applications may maintain their Session Variables in different ways. When these two kinds of applications are integrated, they may need to "copy" Session Variables so they can be shared between them. Logi Session Variables are accessed via @Session tokens, while standard Java application Session Variables are accessed via JSP or, in a Java program, via `javax.servlet.http.HttpSession`.

The **Java Session Copying** element, which is ignored in .NET Logi applications, can be added to the _Settings definition to enable Session Variable copying between the two kinds of Java applications. It has four attributes that let you control which variables are copied, which optimizes performance. The "Include" attributes are processed first, then the "Exclude" attributes. The attributes include:

Attribute	Description
Copy From Java Exclude	Specifies a comma-separated list of regular expressions that identify the Java application session variables that *will not* be copied to the Logi Info application session space. Do not use session variable names that contain spaces.
Copy From Java Include	Specifies a comma-separated list of regular expressions that identify the Java application session variables that *will* be copied to the Logi Info application session space. Do not use session variable names that contain spaces.
Copy To Java Exclude	Specifies a comma-separated list of regular expressions that identify the Logi Info application session variables that *will not* be copied to the Java application session space. Do not use session variable names that contain spaces.
Copy From	Specifies a comma-separated list of regular expressions that identify the Logi Info application session variables

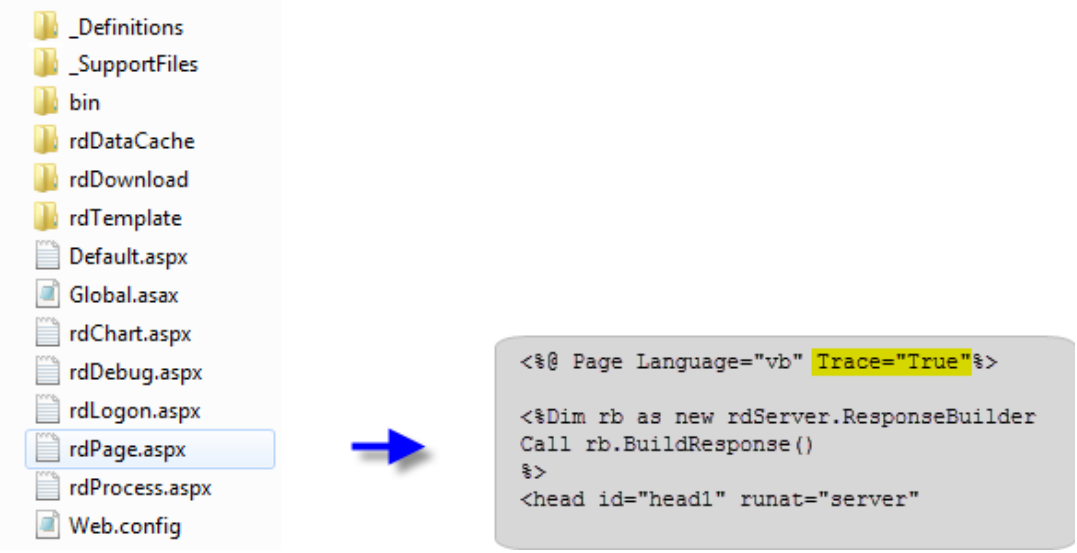
Attribute	Description
Java Include	that *will* be copied to the Java application session space. Do not use session variable names that contain spaces.

It is possible to have almost all session variables copied automatically, but that typically doesn't produce the best performance. However, if you want to use this behavior, create element source code like the example below and paste it into your _Settings definition XML code:

```
<JavaSessionCopying CopyToJavaInclude="myVar1, myVar2"
 CopyFromJavaInclude="myVar3, myVar4" />
```

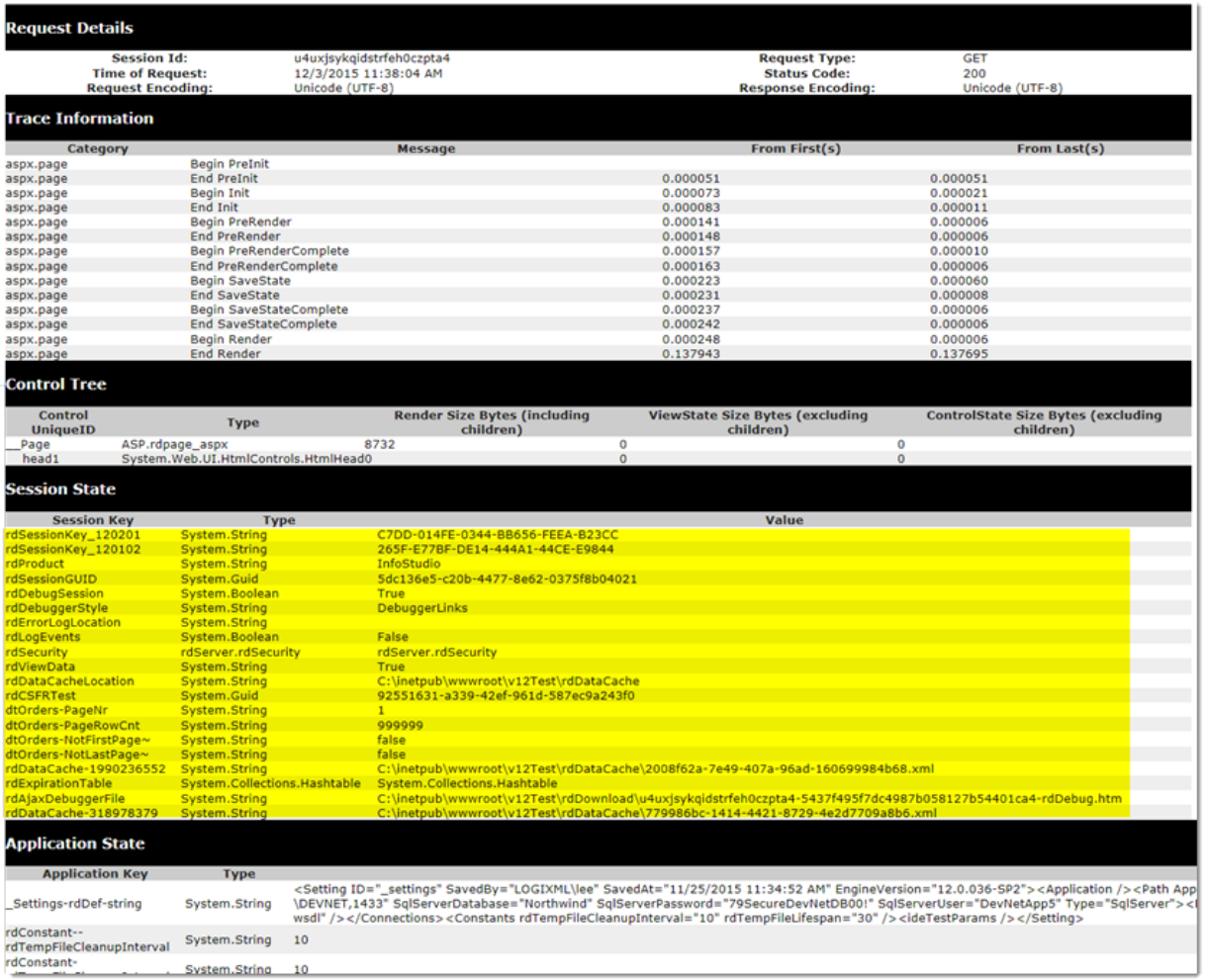
Viewing Session Variables in Debugger

The Logi **Debugger Trace Page** is very useful for identifying problems and includes helpful information, such as the values of all Request variables. However, it doesn't usually display SessionVariables. However, for .NET applications, you can enable a *full trace page* that will show you the Session Variables and much more.



To turn on full tracing:

1. Locate your Logi application's `rdPage.aspx`file in the application folder, as shown above.
2. Use a text editor, such as Notepad, to edit the file to include `Trace="True"`, as shown above.



Request Details

Session Id:	u4uxjsykqidstrfeh0czpta4	**Request Type:**	GET
Time of Request:	12/3/2015 11:38:04 AM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.000051	0.000051
aspx.page	Begin Init	0.000073	0.000021
aspx.page	End Init	0.000083	0.000011
aspx.page	Begin PreRender	0.000141	0.000006
aspx.page	End PreRender	0.000148	0.000006
aspx.page	Begin PreRenderComplete	0.000157	0.000010
aspx.page	End PreRenderComplete	0.000163	0.000006
aspx.page	Begin SaveState	0.000223	0.000060
aspx.page	End SaveState	0.000231	0.000008
aspx.page	Begin SaveStateComplete	0.000237	0.000006
aspx.page	End SaveStateComplete	0.000242	0.000006
aspx.page	Begin Render	0.000248	0.000006
aspx.page	End Render	0.137943	0.137695

Control Tree

Control UniqueID	Type	Render Size Bytes (including children)	ViewState Size Bytes (excluding children)	ControlState Size Bytes (excluding children)
__Page	ASP.rdpage_aspx	8732	0	0
head1	System.Web.UI.HtmlControls.HtmlHead	0	0	

Session State

Session Key	Type	Value
rdSessionKey_120201	System.String	C7DD-014FE-0344-BB656-FEEA-B23CC
rdSessionKey_120102	System.String	265F-E77BF-DE14-444A1-44CE-E9844
rdProduct	System.String	InfoStudio
rdSessionGUID	System.Guid	5dc136e5-c20b-4477-8e62-0375f8b04021
rdDebugSession	System.Boolean	True
rdDebuggerStyle	System.String	DebuggerLinks
rdErrorLogLocation	System.String	
rdLogEvents	System.Boolean	False
rdSecurity	rdServer.rdSecurity	rdServer.rdSecurity
rdViewData	System.String	True
rdDataCacheLocation	System.String	C:\inetpub\wwwroot\v12Test\rdDataCache
rdCSFRTest	System.Guid	92551631-a339-42ef-961d-587ec9a243f0
dtOrders-PageNr	System.String	1
dtOrders-PageRowCnt	System.String	999999
dtOrders-NotFirstPage~	System.String	false
dtOrders-NotLastPage~	System.String	false
rdDataCache-1990236552	System.String	C:\inetpub\wwwroot\v12Test\rdDataCache\2008f62a-7e49-407a-96ad-160699984b68.xml
rdExpirationTable	System.Collections.Hashtable	System.Collections.Hashtable
rdAjaxDebuggerFile	System.String	C:\inetpub\wwwroot\v12Test\rdDownload\u4uxjsykqidstrfeh0czpta4-5437f495f7dc4987b058127b54401ca4-rdDebug.htm
rdDataCache-318978379	System.String	C:\inetpub\wwwroot\v12Test\rdDataCache\779986bc-1414-4421-8729-4e2d7709a8b6.xml

Application State

Application Key	Type	
_Settings-rdDef-string	System.String	<Setting ID="_settings" SavedBy="LOGIXML\lee" SavedAt="11/25/2015 11:34:52 AM" EngineVersion="12.0.036-SP2"><Application /><Path App \DEVNET,1433" SqlServerDatabase="Northwind" SqlServerPassword="79SecureDevNetDB00!" SqlServerUser="DevNetApp5" Type="SqlServer"><(wsdl" /></Connections><Constants rdTempFileCleanupInterval="10" rdTempFileLifespan="30" /><ideTestParams /></Setting>
rdConstant--rdTempFileCleanupInterval	System.String	10
rdConstant-	System.String	10

The full trace page, similar to the one shown above, will now appear at the bottom of every report page and you can see the Session State section, highlighted, showing all the Session Variables. You'll see quite a few that start with "rd" - these are created by the Logi Engine.

Managing Session Timeout

The Session Timeout element makes it easy to prevent errors caused by session timeouts.

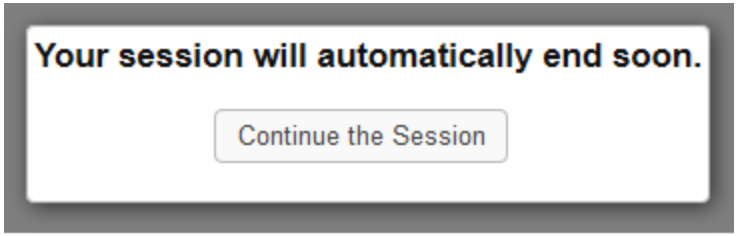
The following topics discuss use of the Session Timeout element:

- Session Timeout Attributes
- Using the Session Timeout Element

About the Session Timeout Element

When a user's working with a Logi Info application and leaves the browser session inactive for some time, the server-side session may expire and end. This can cause problems with pages that need the session state to be live. The **Session Timeout** element makes it easy to prevent errors caused by session timeouts.

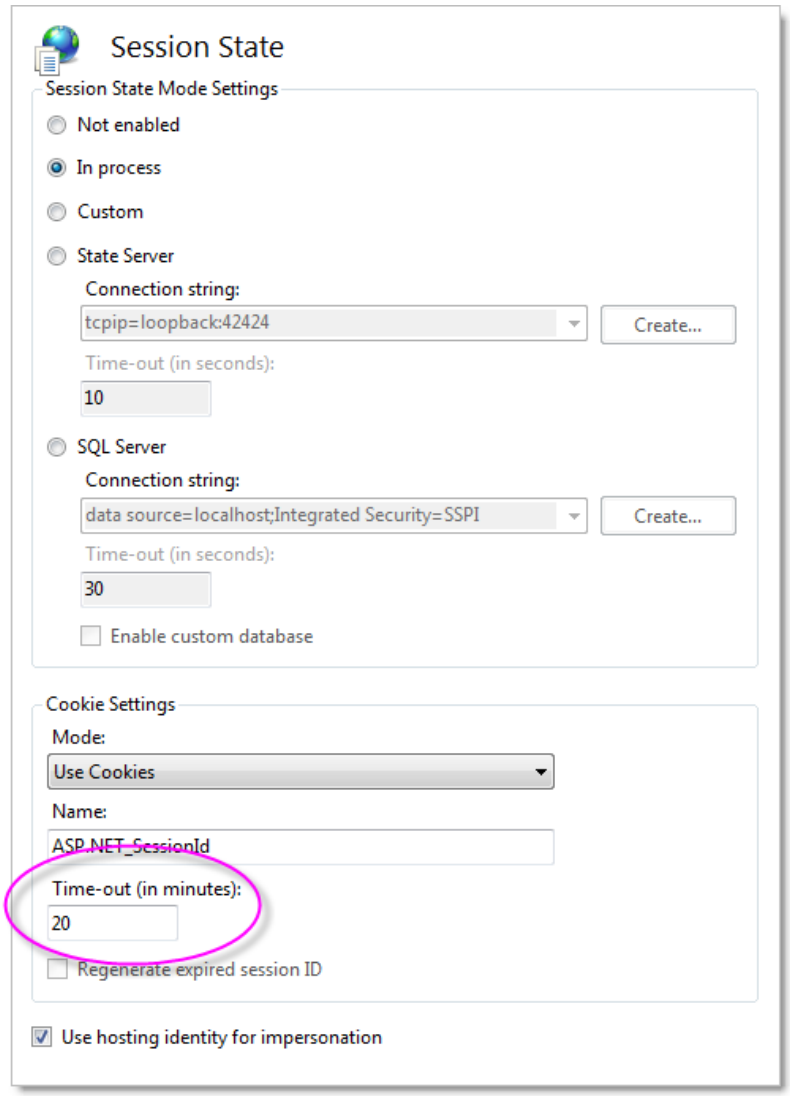
The element operates in two modes, depending on the value of its **Session Auto Keep Alive** attribute. When set to *True*, the session remains alive for as long as the browser is open. This is accomplished by automatically "pinging" the server from the browser, thus informing the server that the browser is still open. Once the browser closes, the server-side session times-out after its expiration period.



If the **Session Auto Keep Alive** attribute is set to *False*(the default), the user will be prompted with a pop-up panel in the browser, which asks if the session should be kept alive. This prompt appears some minutes before the timeout occurs. If the user confirms the prompt, the session is kept alive and the user may continue working. If the pop-up is ignored for some minutes, the browser session is redirected to another web page, such as a login page.

Web Server Session Expiration Setting

The Session Timeout element *does not* control or affect the web server's session expiration setting. That depends entirely on the web server's configuration.



For Windows IIS, the default expiration setting is 20 minutes and can be set using either the IIS Management Console (under the Logi application virtual directory → Session State (a typical configuration for IIS 7.5 is shown above) or in the application's

`Web.config` file. Exact settings will vary depending on which Session State mode is being used. Refer to Microsoft's documentation on the subject for more information. For Linux/UNIX servers, timeout configuration locations vary; consult the documentation. For example, for Apache Tomcat 8 servers, the value can be found in:

```
C:\Program Files\Apache Software Foundation\Tomcat 8.0\webapps\yourLogiApp\WEB-INF\web.xml
```

as this code:

```
<session-config>
<session-timeout>20</session-timeout>
</session-config>
```

where the value is in minutes.

Session Timeout Attributes

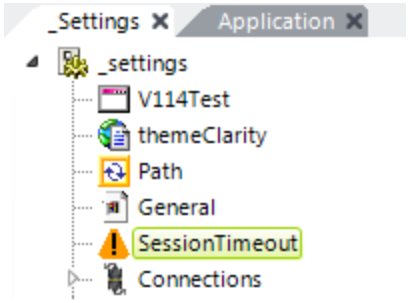
The Session Timeout element has the following attributes:

Attribute	Description
Session Auto Keep Alive	Specifies the operating mode. When set to *True*, the session remains alive for as long as the browser is open. Once the browser closes, the server-side session times-out after its expiration period. If set to *False* (the default), the user will be asked, in a warning pop-up panel, if they want to continue the session. This prompt appears some minutes before the timeout occurs. If the user confirms the prompt, the session is kept alive. If the pop-up is ignored for some minutes, the browser session is redirected to another web page, such as a login page.
Session Ended URL	Specifies a URL that the browser will redirect to once the session ends. The default is the standard Logi logout page `rdLogout.aspx`.
Session Keep Alive Caption	Specifies the button caption in the warning pop-up panel. Clicking the button keeps the session alive. The default caption is "Continue the Session".
Session Keep Alive Caption Class	Specifies the class used to style the Session Keep Alive Caption text.
Session	Specifies the text displayed in the warning pop-up panel. The default value is "Your session will automatically

Attribute	Description
Warning Caption	end soon".
Session Warning Caption Class	Specifies the class used to style the Session Warning Caption text.
Session Warning Duration	Specifies the amount of time, in minutes, that the warning pop-up panel will be displayed before the browser is redirected to the Session Ended URL value. The default value is 3 minutes. This formula is used to determine when the warning pop-up panel will be displayed: `Server session expiration - (Session Warning Duration + 2)` . For example, if the server session expiration setting = 20 minutes and Session Warning Duration = 3 minutes, the warning pop-up will appear after 15 minutes of inactivity.
Template Modifier File	Specifies the name of an optional Template Modifier file. The template modifier file can be in any folder accessible to the application; if a fully-qualified file path is not provided then the application expects it to be in the _SupportFiles folder.

Using the Session Timeout Element

The Session Timeout element is easy to implement:



Simply add the Session Timeout element in your _Settings definition, as shown above, then set its optional attributes as desired.

Load Balancing Configuration

Load balancing is a technique used to distribute workload evenly across two or more web servers in order to optimize resource utilization, maximize throughput, minimize response time, and avoid overload. This topic discusses the configuration requirements for using Logi applications in load-balanced .NET and Java environments. If you need information about sizing, see *System Requirements - Logi Info*.

The following topics discuss processes that you will use for two types of load balancing scenarios, sticky, and non-sticky:

- Sticky Session Load Balancing
- Non-Sticky Session Load Balancing
- Application Resource Centralization

About Load Balancing

In a standard environment, with one server, Logi Info establishes a session with the first HTTP request. That same server handles all subsequent requests. You can use multiple web servers in a "server pool" and control them using load balancing management software. This software monitors the port where external client browsers connect to access services, and distributes requests among the available servers. Replies may be routed back through the management software, hiding the existence of multiple servers. This provides an additional security benefit, by preventing clients from contacting managed servers directly.

You can use two types of load balance session configuration, using sticky sessions and non-sticky sessions. Logi recommends using a sticky session configuration, because it is faster and easier to troubleshoot. The session configuration you use drives how you centralize shared resources. Remember that HTTP sessions are stateless, and that you may have to use cookies or some other tracker to make sure you maintain the proper stickiness when you use HTTP sessions.

Make sure you read the sections of this topic describing Application Resource Centralization, as it applies to both sticky and non-sticky sessions.

Sticky Session Load Balancing

In a load balanced environment with multiple servers, Logi Info establishes sessions based on server availability, and the assigned server handles all subsequent requests. This is called using *session affinity* or "sticky" sessions. Logi Info requires resource consolidation to work in a multi-server deployment.

To deploy a sticky environment, follow these steps:

1. Configure Load Balancer to be set to sticky (session affinity)
2. Prepare environment for centralization of resources (see Application Resource Centralization)
3. Configure Logi Info application
4. Deploy application to all web servers

Non-sticky Session Load Balancing

Non-sticky session configuration means that you do not route all the requests to the same server that handled the initial request. Subsequent requests can be routed to any of the servers in the pool for processing. Again, we recommend you use a sticky session load balancing environment with Logi Info because certain files need to be shared across servers, which can be slower in a network environment.

If you must use a non-sticky session configuration, you will face these challenges:

- Session replication can be contentious
- File access performance can impact app performance and in some cases can cause race-conditions, leading to errors
- Performance impact on distributed file systems, e.g. EFS, would be greater, especially for high file I/O requirements for rdDatacache folder
- Hosting on cloud (AWS, Azure, GCP, etc.) adds additional complexity to the above issues

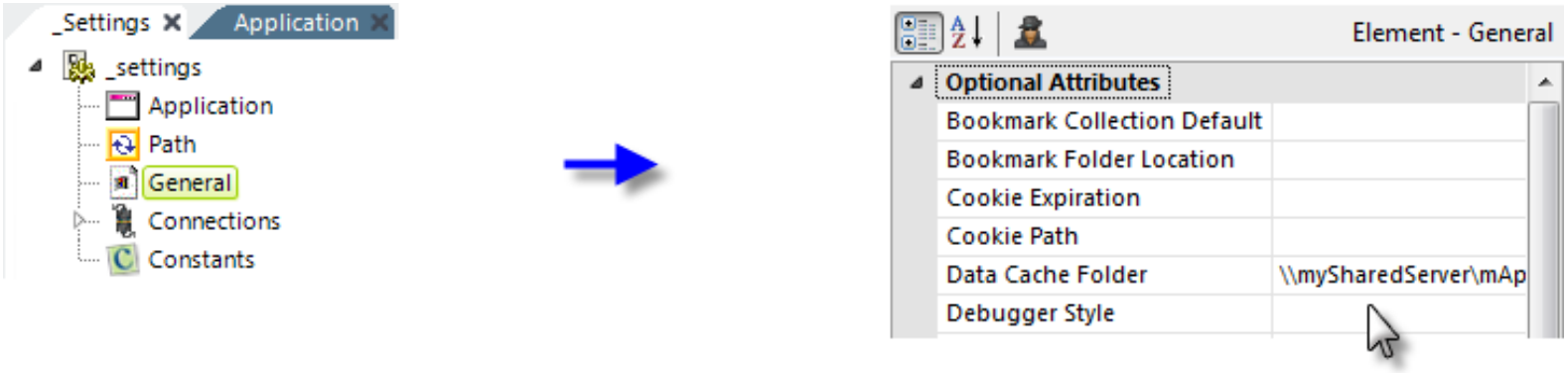
To deploy a non-sticky environment, follow these steps:

1. Configure Load Balancer to be set to non-sticky
2. Prepare environment for centralization of resources (see Application Resource Centralization)
3. Centralize rdDataCache (see below)
4. Configure Logi Info application
5. Deploy application to all web servers

Centralizing rdDataCache for Non-Sticky Sessions Only

The data cache repository is, by default, the `rdDataCache` folder in a Logi application's root folder. In a standalone environment, where all the requests are processed by the same server, this default cache configuration is sufficient.

In a non-sticky load-balanced environment, centralizing the data cache repository is required. This will pose additional challenges to customers who want to use a non-sticky load balanced environment. If you do insist on using this configuration, make sure you centralize the data cache in this way.



This is accomplished in Studio by editing a Logi application's _Settings definition, as shown above. You need to set the **General** element's **Data Cache Location** attribute value to the UNC path of a shared network location accessible by all web servers. You need to make this change in the _Settings definition for each instance of the Logi application (i.e. on each web server). When you specify a network folder for the data cache, you need to change the identity that the web server uses to run the application (usually ASPNET or NETWORK SERVICE or the Application Pool), or impersonates, to a network domain account that has Full Control access permissions for the specified folder.

This attribute will support relative path notation, such as two periods ".." used to indicate a parent folder. For example,

```
@Function.AppPhysicalPath~\..\..\..\..\OurDataCache
```

Managing Session State for Non-Sticky Sessions

A "session" is defined as the period of time in which a unique user interacts with a particular web application. HTTP is a "stateless" protocol, in the sense that a web server is concerned only with the current HTTP request for any given web page. The web server retains no knowledge of previous requests and the stateless nature of HTTP requests presents unique challenges when writing web applications. In clustered environments, where non-sticky sessions are desired, session state needs to be replicated to all members of the cluster.

State Replication for ASP.NET and IIS for Non-Sticky Session Environments

Your ASP.NET applications require the session state to be maintained using one of these three session management options:

- **InProc** - Session state is stored locally on the web server and is managed in the same worker process as the ASP.NET application.
- **StateServer** - Session state is managed by the ASP.NET state service, which runs outside of the ASP.NET worker process. The service can run on a different server to support web farms.
- **SQLServer** - Session state is managed outside of the ASP.NET worker process and is stored in a SQL Server database. Like the StateServer option, this method can support web farms.

 Additional considerations, other than load balancing, may affect the choice of session state management options. IIS is configured by default to manage session information using the "InProc" option. For both standalone and load-balanced, sticky environments, this option allows a single server to manage the session information for the life of the session. For non-sticky, load-balanced configurations, session state needs to be managed centrally. Since requests can be processed by any of the available

servers, the servers need to be able to access session information from a common location. Therefore, you should use either the StateServer or SQLServer management options in this circumstance. Use your IIS Management Tool, in the ASP.NET configuration tab or dialog box, to actually configure your session management options; however, the details are beyond the scope of this document. Refer to your IIS documentation.

State Replication for Java and Apache Tomcat for Non-Sticky Session Environments

Use this link for detailed instructions about creating clustered implementations of Apache Tomcat. This procedure describes how you configure Tomcat for in-memory (multicast) and database (persistent) session replication across all members of a cluster (some of the information presented here is drawn from the clustering instructions at the link above). To configure *in-memory* session replication:

1. Modify your `server.xml` file by adding the following in either the `<Engine>` or `<Host>` sections:

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```

To configure use of a *database* for session replication, follow these steps:

1. Ensure that the `<Cluster>` element is disabled in your `server.xml` file to turn off in-memory replication.
2. Navigate to the root of your Logi app and, if it doesn't already exist, create a folder named `META-INF`.
3. In the `META-INF` folder create a file named `context.xml` and open it in a file editor.
4. In the file, define a root `<Context>` element, and then add the following `<Manager>` element to that section:

```
<Manager className="org.apache.catalina.session.PersistentManager" >
```

5. Determine what type of database you want to use for session persistence and copy the `.JAR` driver file for that database into the Tomcat `lib` folder.
6. Add the following `<Store>` element within the `<Manager>` section:

```
<Store className="org.apache.catalina.session.JDBCStore"
  connectionURL="jdbc:<your_jdbc_connection_url>"
  driverName="<your_jdbc_driver_classname>"
  sessionIdCol="session_id"
  sessionValidCol="valid_session"
  sessionMaxInactiveCol="max_inactive"
  sessionLastAccessedCol="last_active"
  sessionTable="tomcat_sessions"
  sessionAppCol="app_context"
  sessionDataCol="session_data"
/>
```

7. Execute the following SQL command against the database specified in your JDBC connection URL:

```
create table tomcat_sessions (
  session_id varchar(100) not null primary key,
```

```
  valid_session char(1) not null,
  max_inactive int not null,
  last_access bigint not null,
  app_context varchar(255),
  session_data mediumblob,
  KEY kapp_context (app_context)
);
```

Configure Logi Info Application

Regardless of which replication method you chose above, complete the session configuration with these steps:

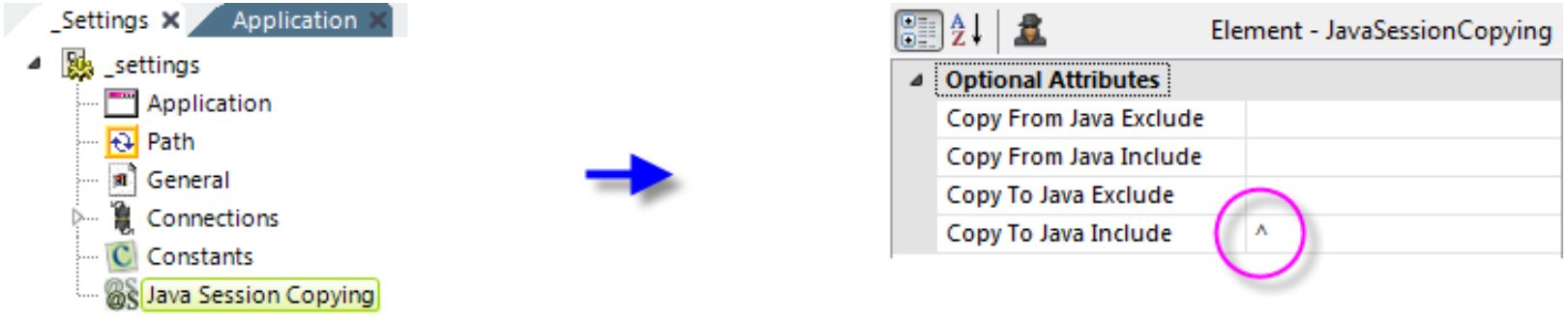
1. In `<LogiAppFolder>/WEB_INF/web.xml`, ensure that the `<distributable/>` tag is included.
2. In `<LogiAppFolder>/WEB_INF/web.xml`, set the **EnableSessionPersistency** context parameter to *True*. This parameter is present in all Logi apps by default.

   Here's a partial file example, with both additions highlighted:.

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd"[]>
    <web-app id="WebApp_ID">
    <display-name>Logi Server for Java 12.2.225-SP5</display-name>
    <description>Logi Server for Java 12.2.225-SP5</description>
    <distributable/>
    <context-param>
```

```
<param-name>EnableSessionPersistency</param-name>
     <param-value>True</param-value>
    </context-param>
...
```

3. In your Logi application's _Settings definition, add a **Java Session Copying** element:



Be sure to set the **Copy to Java Include** attribute value, using a caret symbol "^".

Application Resource Centralization

You will use the same procedures as best practice for both sticky and non-sticky load balancing management. However, for sticky implementation, ensure rdDataCache directory is set to default configuration (under root of application directory).

Centralizing Files

When using sticky sessions, you want to centralize certain files:

- Saved Bookmark files
- Saved Dashboard files
- SecureKey files
- Error Log files

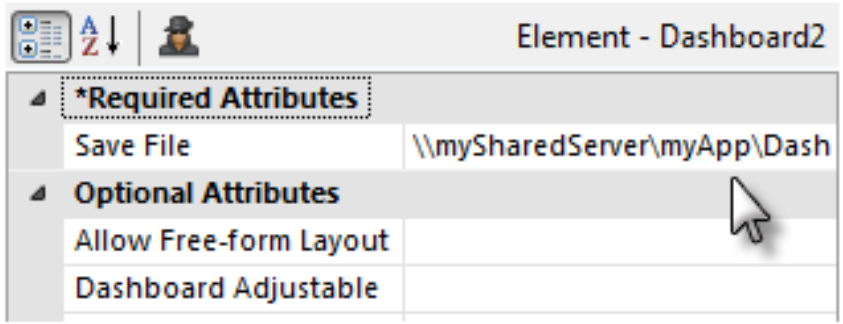
You have options on how you can centralize the Bookmark, Dashboard, and SecureKey files in a database. For more information, see *File To Database Mapping Element*.

 You will also want to centralize your Error log files, but you cannot put them in a database.

Configuring Application

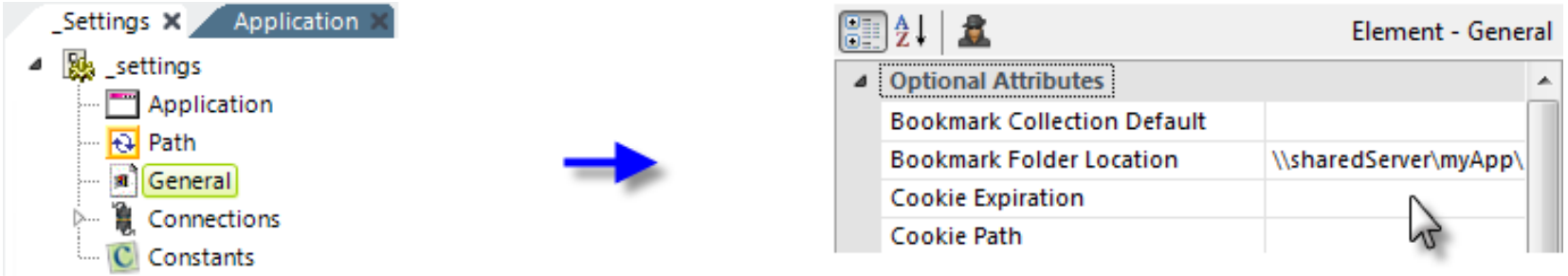
In addition to centralizing the files, as shown above, modify the _Settings file to target the file locations.

"Save" File - Many super-elements, such as the **Dashboard** and **Analysis Grid**, allow the user to save the current configuration to a file for later reuse. The locations of these files are specified in attributes of the elements.



As shown in the Dashboard example above, the **Save File** attribute value should be the UNC path to a shared network location (with file name, if applicable) accessible by all web servers.

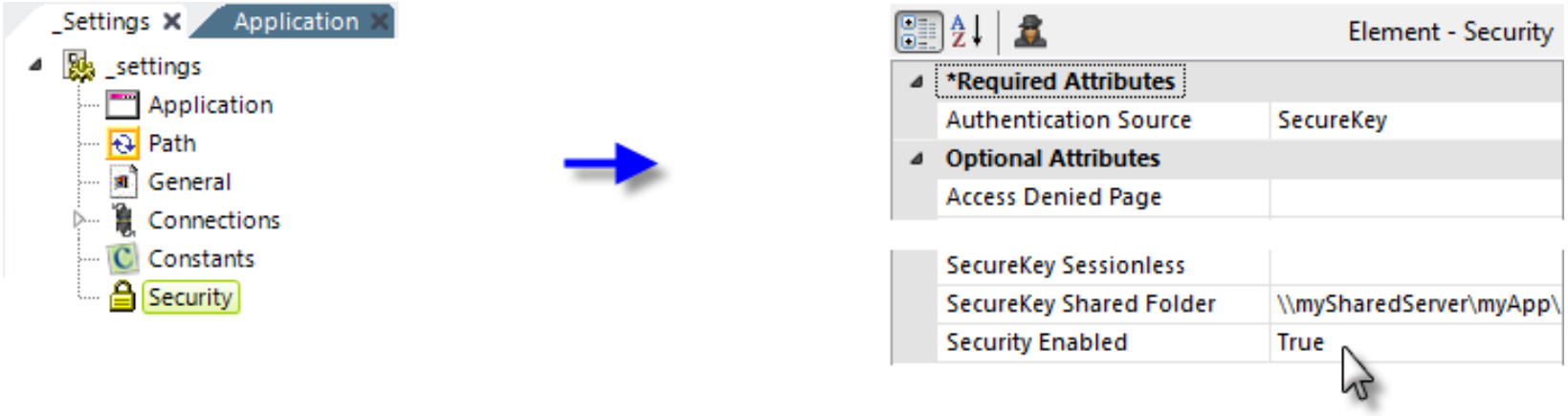
Bookmarks - If used in an application, the location of these files should also be centralized:



As shown above, in the _Settings definition, configure the **General** element's **Bookmark Folder Location** attribute, with a UNC path to a shared network folder accessible by all web servers.

- In general, when you specify a network folder or file to centralize a resource, you need to change the identity that the web server uses to run the application (usually ASPNET or NETWORK SERVICE or the Application Pool), or impersonates, to a network domain account that has Full Control access permissions for the specified folder.
- You can also store your Bookmarks and Save files in a SQL database. See *Bookmarks* for more information.
- You also have options on how you can centralize the Bookmark, Dashboard, and SecureKey files in a database for non-sticky sessions; see *File To Database Mapping Element*.

If you're using Logi SecureKey security in a load-balanced environment, you need to configure security to share requests.



In the _Settings definition, set the **Security** element's **SecureKey Shared Folder** attribute to a network path, as shown above. Files in the SecureKey folder are automatically deleted over time, so do not use this folder to store other files.

Keywords: clustering, cluster, clustered

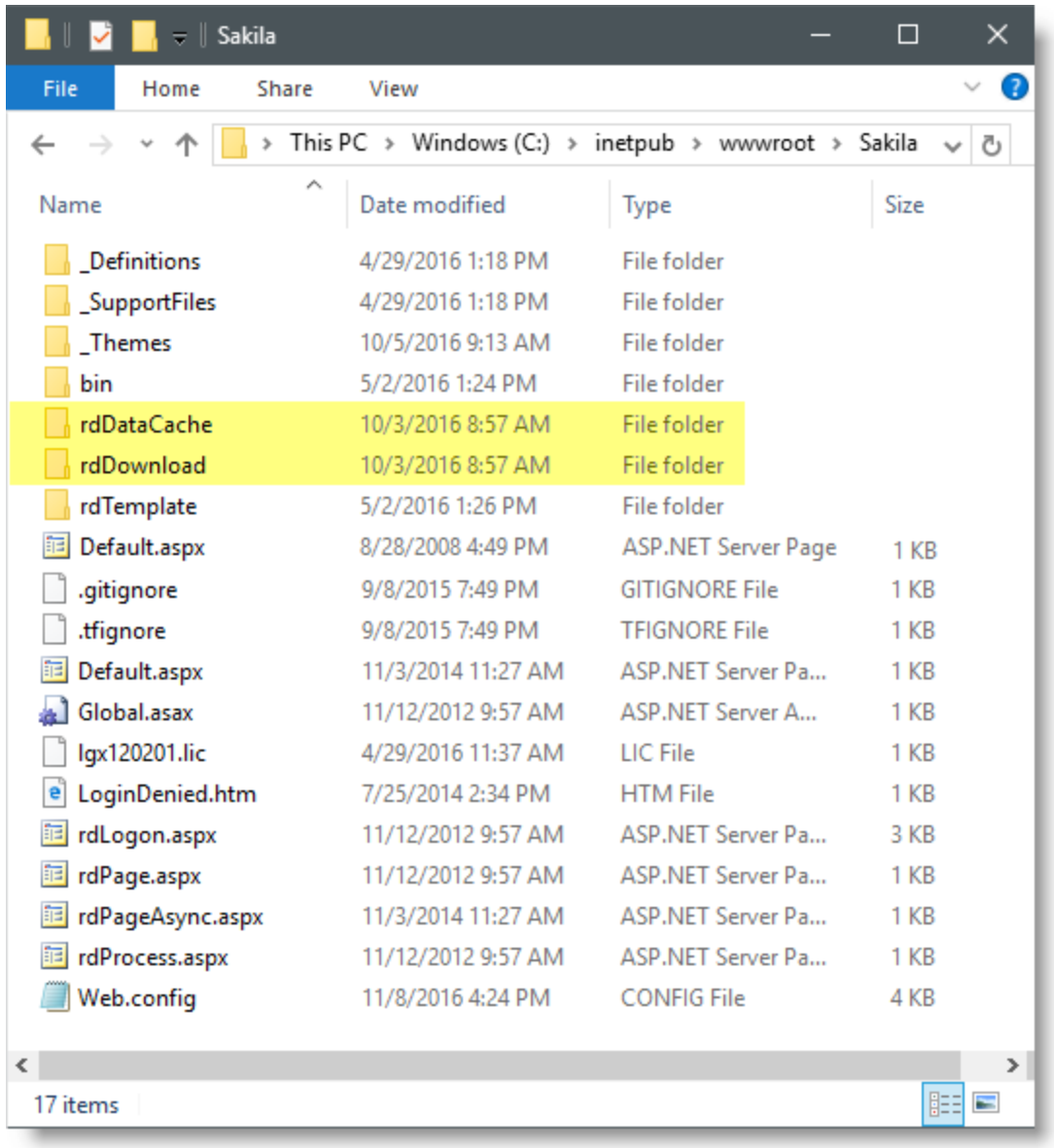
Temporary Cache File Management

The files that make up a Logi application are contained in a single application root folder that has a number of standard sub-folders. Many of these sub-folders have specific purposes; two of the most interesting standard sub-folders are **rdDataCache** and **rdDownload**. These folders hold temporary files for caching data and other purposes; they are managed automatically.

The following topics discuss the temporary cache file management folders:

About Temporary Cache Files

As you may know, physically a Logi application consists of a single folder that contains a number of standard sub-folders and files. Many of these sub-folders have specific purposes. For example, .css files placed in the *_SupportFiles* folder will appear in Logi Studio automatically and will be included in selection lists for assigning style sheets.


Sakila
File
Home
Share
View
This PC › Windows (C:) › inetpub › wwwroot › Sakila
Name
Date modified
Type
Size
_Definitions 4/29/2016 1:18 PM File folder
_SupportFiles 4/29/2016 1:18 PM File folder
_Themes 10/5/2016 9:13 AM File folder
bin 5/2/2016 1:24 PM File folder
rdDataCache 10/3/2016 8:57 AM File folder
rdDownload 10/3/2016 8:57 AM File folder
rdTemplate 5/2/2016 1:26 PM File folder
Default.aspx 8/28/2008 4:49 PM ASP.NET Server Page 1 KB
.gitignore 9/8/2015 7:49 PM GITIGNORE File 1 KB
.tfignore 9/8/2015 7:49 PM TFIGNORE File 1 KB
Default.aspx 11/3/2014 11:27 AM ASP.NET Server Pa... 1 KB
Global.asax 11/12/2012 9:57 AM ASP.NET Server A... 1 KB
lgx120201.lic 4/29/2016 11:37 AM LIC File 1 KB
LoginDenied.htm 7/25/2014 2:34 PM HTM File 1 KB
rdLogon.aspx 11/12/2012 9:57 AM ASP.NET Server Pa... 3 KB
rdPage.aspx 11/12/2012 9:57 AM ASP.NET Server Pa... 1 KB
rdPageAsync.aspx 11/3/2014 11:27 AM ASP.NET Server Pa... 1 KB
rdProcess.aspx 11/12/2012 9:57 AM ASP.NET Server Pa... 1 KB
Web.config 11/8/2016 4:24 PM CONFIG File 4 KB
17 items

Two of the most interesting standard sub-folders are *rdDataCache* and *rdDownload*; they have a mechanism associated with them that makes web server storage maintenance much easier.

The rdDataCache Folder

When data is returned from a data source, the Logi Server Engine may cache it in server memory or in temporary XML files or in both. When certain charts are used, temporary code fragment files are generated. These and other temporary files are created by the system in the rdDataCache folder.

When data is cached in this folder it makes easier to "page" through the data and to retrieve it quickly. However, if a new data query is executed, then the data cached here is replaced. The two exceptions to this are when **DataLayer.Cached** is used and when a super-element such as the **Analysis Grid**, which is designed to work with cached data, are used. Both of these exceptions have specific mechanisms for refreshing their data.

The default name and location of the rdDataCache folder works very well for most implementations but they can be changed in *The _Settings Definition*. This might be helpful when very large data sets and large numbers of user raise concerns about temporary files consuming all of the free disk space, or in clustered server environments where sharing is required. These are unusual situations, however, and the default is appropriate for most use-cases.

rdDownload Folder

The rdDownload folder is the home of a number of other temporary files. For example, when debugging links are turned on and trace pages are created, the XML debug files are stored here. *Uploaded* files, oddly enough, temporarily reside here, too, before being relocated, and exports also use this folder for temporary files.

There is no way to rename or relocate this folder.

Developers who need to store data in temporary files are free to use these folders, too (traditionally, rdDownload). And it's a very good idea to do so. The server account under which Logi applications run typically has been granted Write permissions to these folders, so files can be written there.

Securing rdDownload

When reports or data are exported from Report definitions, the rdDownload folder temporarily contains exported PDF, Excel, etc. files in order to make them available for download by the user who created them. By default, these files have names based on GUIDs, so directly accessing them without permission by guessing their names *is* possible but unlikely. However, if you'd like to further secure these files, you can engage two special features.

- The first feature requires that the Session ID of the user requesting an export file download match that of the user who created the export file, otherwise an HTTP 403 error is returned.
- The second feature ensures that supporting files used in the process of creating an export, and saved in rdDownload, are deleted once the export file is created but before it's streamed back to the requesting user.

To enable these features, you must manually alter the Logi application's `web.config`file, in the application's root folder. The file already contains the code required and all you have to do is uncomment it to enable it. Here's the code:

```
     <!-- Uncomment the following rdDownloadSecPageHandler verb in the
 httpHandlers section to enable the automatic deletion of export files in
 the rdDownload directory. This also enables additional security for these
 files.  -->
     <system.web>
<!--
         <httpHandlers>
                 <add
 verb="*" path="rdDownload/*"
 type="rdServer.rdDownloadSecPageHandler,rdServer"/>

         </httpHandlers>
-->
     </system.web>
     <!--  Uncomment the  rdDownloadSecPageHandler in the
 system.webServer section to enable rdDownload directory Security
   -->
     <system.webServer>
<!--
           <validation
 validateIntegratedModeConfiguration="false"/>
               <handlers>
```

```
    <add  verb="*"
 path="rdDownload/*" name="rdDownloadSecPageHandler"
 type="rdServer.rdDownloadSecPageHandler,rdServer"
 resourceType="Unspecified"/>
            </handlers>
-->
     </system.webServer>
```

These features *do not* apply to exports created using Process Tasks.

Without these feature, these files are cleaned up periodically - see "Temporary File Clean Up" on the next page.

Temporary File Clean Up

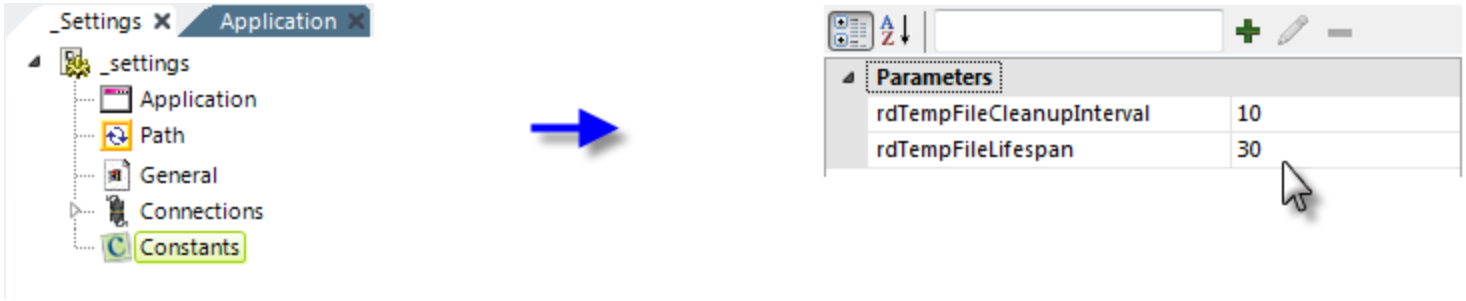
When you create temporary files, you run the risk of consuming all of your hard disk storage space. To manage this, Info includes an automatic mechanism that "cleans up" files in temporary file folders by deleting them.

- Temporary file clean up runs on the very first application page request.
- With each subsequent request, Logi Info checks to see if it's been 5 minutes (default value, configurable--see below) since the last clean up. If so, then clean up is run again.
- When clean up runs, Info deletes any files older than 60 minutes (default value, configurable--see below) that are found in the rdDataCache and rdDownload folders, and the SecureKey Shared Folder, if that type of Logi Security is being used.
- When clean up runs, a message is added to the debug log.

In addition, when the web server is shut down cleanly, Info deletes all the files in rdDataCache and rdDownload automatically.

Configure the Time Intervals

The two time intervals mentioned above are the default values but they can be configured differently.



As shown above, two constants can be added to the _Settings definition to configure the clean up mechanism.

- `rdTempFileCleanupInterval` - The time, in minutes, between clean ups. If this value is -1, clean up will not run at all; if the value is 0, clean up will run with *every* page request. Default value: *5 minutes*.
- `rdTempFileLifespan` - The minimum file age, in minutes, before a file qualifies to be deleted when the clean up runs. Default value: *60 minutes*. Your web server's **Session Timeout** setting should *not* be configured to be *less* than the value assigned to `rdTempFileCleanupInterval` or its default value of 5 minutes.

For information about these and other constants settings, see *Special Constants* and *Reserved Words*.

Doctype Declarations

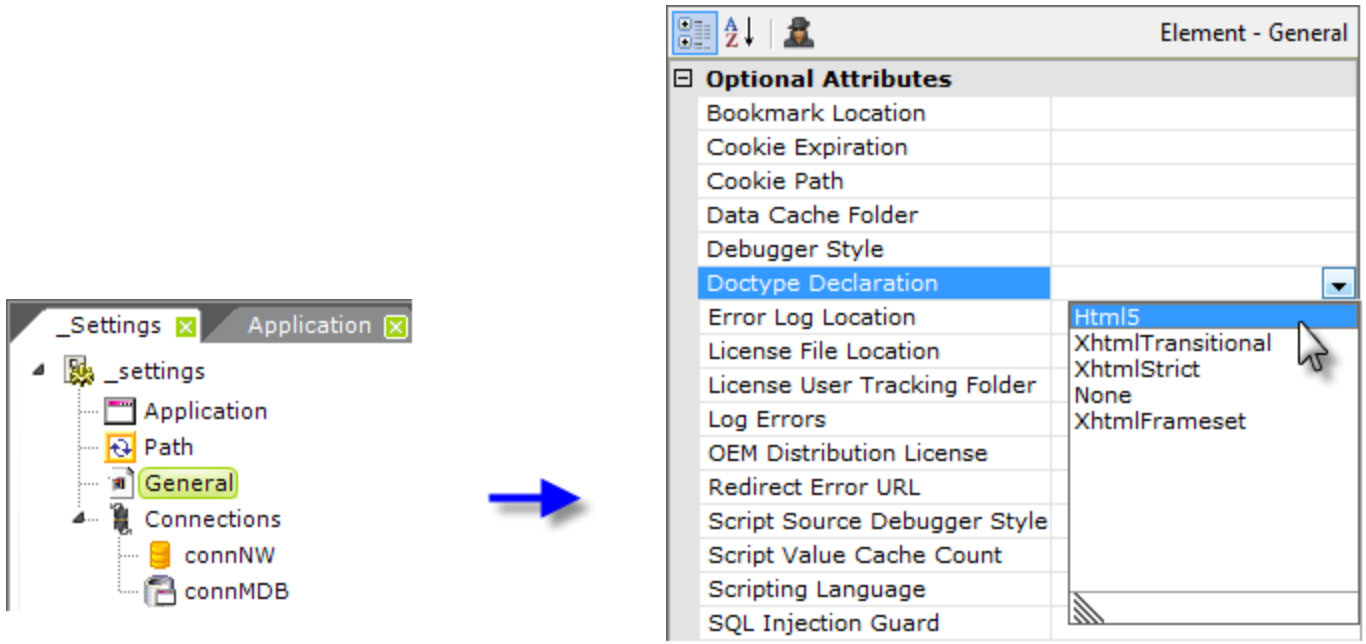
Developers can set the "document type declaration" or "doctype" of their Logi applications. Though it appears as the very first thing in an HTML/XHTML page, the doctype is not an HTML tag; instead it's an instruction to the web browser that identifies the version of the markup language the page is written in.

The following topics discuss the doctype options in Logi products:

- HTML5
- XHTML Strict
- XHTML Transitional
- XHTML Frameset
- What's the Impact?

About Doctypes

The HTML standards recognize a "document type declaration" (or "doctype") which is an instruction to the web browser that indicates which version of the markup language the page is written in. The doctype is a key component of standards-compliant web pages. Doctype tells modern browsers how they should apply HTML and CSS standards in rendering a page. These include more recent versions of Firefox, Chrome, Internet Explorer, Opera, and Safari. The doctype may include a URL reference to a Document Type Definition (DTD). The DTD specifies the rules for the markup language, so that browsers can render the content correctly. If an invalid, or no, doctype is specified in an HTML page, browsers go into "quirks" mode, where they each handle the markup in different and, possibly, outdated ways.



In Logi Studio, as shown above, the doctype is configured using the **Doctype Declaration** attribute of the **General** element in the _Settings definition. Five options are available using the pull-down selection list. **HTML5**isthe*default* value if no DocType Declaration value is selected or entered. If *None* is selected, no doctype instruction is added to the HTML pages generated by the Logi Server Engine. The other four options are discussed individually in the following sections.

HTML5

When this option is selected for the Doctype Declaration attribute, the following is placed at the beginning of the generated HTML page: `<!DOCTYPE HTML>` This declares the document to be **HTML5**, which is not based on SGML and therefore does not require a reference to a DTD like other doctype declarations. It's the fifth revision of the HTML standard and its core aims have been to improve the language, adding support for the latest multimedia content, while keeping code easily readable by humans and consistently understood by computers and devices. HTML5 is intended to subsume not only HTML 4, but also XHTML 1 and DOM Level 2 HTML.

XHTML Strict

When this option is selected for the Doctype Declaration attribute, the following is placed at the beginning of the generated HTML page: `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">` This declares the document to be the XML version of **HTML 4.01 Strict**. This is a trimmed-down version of HTML 4.01 that emphasizes structure over presentation. Deprecated elements (such as Font) and attributes (including most presentational attributes), frames, and link targets are not allowed in HTML 4 Strict. Style can be applied only through style sheets. By writing to HTML 4 Strict, developers can achieve accessible, structurally-rich reports that easily adapt to style sheets and different browsing situations. However, HTML 4 Strict documents may look bland on very old browsers that lack support for style sheets.

XHTML Transitional

When this option is selected for the Doctype Declaration attribute, the following is placed at the beginning of the generated HTML page: `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">` This declares the document to be the XML version of **HTML 4.01 Transitional**. HTML 4 Transitional includes all of the elements and attributes of HTML 4 Strict and adds presentational attributes, deprecated elements, and link targets. For example, style tags can be included right in the HTML.

XHTML Frameset

When the Doctype Declaration attribute value is left blank, this is the default option. When this option is in effect or selected, the following is placed at the beginning of the generated HTML page: `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">` This declares the document to be the XML version of **HTML 4.01 Frameset**. HTML 4 Frameset includes all of the elements and attributes of HTML 4 Transitional and adds support for HTML frames. This is the most "generous" doctype and supports the most features in an HTML document.

What's the Impact?

So, how does the choice of a particular doctype impact a Logi application? It can have quite an impact. *XHTML Frameset* provides a "loose" standard that encompasses everything, and the HTML generated by the Logi Server Engine did not generally cause any DTD-related problems. The default option, *HTML5,* may result in significant rendering differences in different browsers. And, if you select *None*, meaning use no doctype, you will almost certainly see presentation (CSS) differences between browsers. Things won't always be aligned correctly, and style may not be applied where or how you expect it, with the most problems occurring in Internet Explorer. And, in complimentary action, if you *are* seeing differences in the way different browsers present your report pages, the use of a doctype may fix that.


You should be aware that, when you **upgrade** an application to Logi v12, and then open and edit legacy report definitions, Studio will automatically insert the default doctype, *HTML5*, depending on Logi version, into the definitions. This will often cause CSS to work differently and the resulting reports may look different. Developers who don't want to tweak their reports to "correct" these effects, can set the doctype to *None* in order to return the reports to their original appearance. Prior to HTML5, standards evangelists called for developers to use the *XHTML Strict* doctype whenever possible. This "tight" standard requires that you code everything very correctly. For example, if you use **Event Handler** with **Action.Javascript** or **Action.Link** and **Target.Link** elements (earlier versions) in your Logi app, and add JavaScript, you must use the correct case-sensitive spelling for any JavaScript functions there. This means "getElementById()" will work, but "GetElementByID()" will not. Your choice of doctype will also affect any HTML documents that you choose to *embed* in your Logi application. In this case, because the HTML is not being generated by the Log Server Engine, you need to take care to ensure that the coding is compliant with the doctype you've selected. For example, if your embedded HTML document includes the <font> tag and you select the *Strict* doctype, the tag will be ignored because that doctype does not support presentation-related tags within the HTML. If you are new to HTML5 and wish to see which of its features are available with which browsers, see this website: http://www.CanIUse.com.

Style Sheets

Cascading Style Sheets (CSS) provide a class-based mechanism for controlling the appearance of Logi application pages and can eliminate the need for inline formatting and some organizational structures.

The following topics assume that you've already created a basic application whose appearance you want to affect by using a style sheet:

- Cascading Style Sheets for Newbies
- Including a Style Sheet in Your Application
- Editing a Style Sheet in the Studio Workspace
- Editing a Style Sheet in the Style Sheet Class Selector
- Editing a Style Sheet Using an External Style Sheet Editor
- Assigning a Style Sheet to Your Definition
- Assigning Classes to Elements

Cascading Style Sheets for Newbies

If you're a developer with web design experience, you should already be familiar with style sheets and you can skip this topic.

Logi Info includes a feature called **Themes** which can instantly apply pre-built styling to your application. A number of stock themes are included with Logi Info and you can also create your own. For more information about themes, see "Themes" on page 235.

Cascading Style Sheets (CSS) is a technology that allows the presentation aspects of web pages to be separated from the page content. It can be used to add "styling" (e.g. apply fonts, colors, alignment, spacing, and more) to web pages. It's a mature standard, fully supported by the W3C. All modern browsers (Chrome, IE, Edge, Firefox, Safari) support CSS3 and earlier standards.

IE has lagged behind Firefox and Chrome in supporting new CSS standards. Not only does this introduce compatibility issues you may have to deal with but, because Logi Studio uses the Windows operating system's IE components, Studio's Preview function may be affected. Issues of this nature tend to arise especially if you integrate third party libraries that make heavy use of HTML5 and CSS3 into your Logi application.

As used with Logi applications, style sheets are independent text files that define style *classes*; with property settings. For example, the definition for the class "font9ptBold", shown below, contains *selectors* that cause text to appear in an 8-point, bold font:

```
.font9ptBold {
 font-size: 8pt;
 font-weight: bold;
}
```

Classes are assigned to elements within a Logi application in order to control their appearance. For example, we might assign the font9ptBold class to a Label element so that its caption will appear on the web page in an 8-point, bold font. CSS includes a wide variety of selectors that can affect everything from font weight, size, and color to the positioning of images on the page.

Use of style sheets creates slightly more complexity for the developer than having appearance properties directly associated (as attributes) with elements in Logi Studio. The trade-off, however, is that they offer far more flexibility and browser-version compatibility.

Logi Info is CSS standards-compliant and, to help you work with style classes, Logi Studio includes a style sheet editor. Logi Info allows you to set a "doctype declaration" for your application, which gives the browser some guidance about how to render the page. This is discussed in "Doctype Declarations" on page 174 and can help sort out some the behavioral differences between browsers.

The "cascading" part of the technology's name refers to the way in which the effects of classes permeate *downward* through hierarchical structures of elements in web pages. The effects start at the top and, unless superceded by other classes, continues downward within the container, structure, or child elements.

You can develop your own "standard" style sheet file with a .css file extension, and then *re-use it* or parts of it with different Logi applications. This saves development time and promotes a consistent look for your applications, if desired. You may care to develop and share a "company style sheet" so that all of your Logi application developers produce pages with the same appearance.

Style classes can be applied in a number of ways, including "inline" or embedded directly into HTML. However, it's beyond the scope of this topic to provide a complete explanation of Cascading Style Sheets; excellent resources can be found online.

Logi products include a feature called **Themes** which can instantly apply style and other presentation touches to your report. A number of stock themes are included with the products and you can also create your own. For more information about themes, see "Themes" on page 235.

Create an Example Style Sheet

Select and copy the following text, paste it into Notepad or a similar text editor, and save it to your desktop (or any other location you prefer) as `CSSExample.css`:

```
/* CSSExample.css: our example style sheet */
.myTable {
 padding: 5px;
border-width: 2px;
border-style: solid;
border-color: Silver;
border-collapse: collapse;
background-color: White;
font-family: Verdana, Arial; /* if Verdana font not found, will use Arial */
font-size: 9pt;
font-weight: normal;
color: Black; /* this is the font color */
}

.myTable TH {/* TH: automatically applied to table header */
padding: 5px;
border-style: solid;
```

```
border-color: Silver;
background-color: #F0F0F0; /* colors can also be specified using hex */
font-size: 10pt;
color: Maroon;
}

.myTable TD {/* TD: automatically applied to table cells */
border: 2px solid Silver; /* combines three border selectors into one */
padding-left: 5px;
padding-right: 5px;
text-align: center;
}

.fontBold {
 font-weight: bold;
}
```

If you're wondering what those "TH" and "TD" designations are, they relate to the tags that make up an HTML table: TH = Table Header, and TD = Table Data. By including them as shown above, their classes will be automatically applied to any Data Table that uses the myTable class. This is a bit of CSS shorthand that saves you time and effort.
The next two sections discuss how to include this style sheet in your application, and how to edit it within Studio.

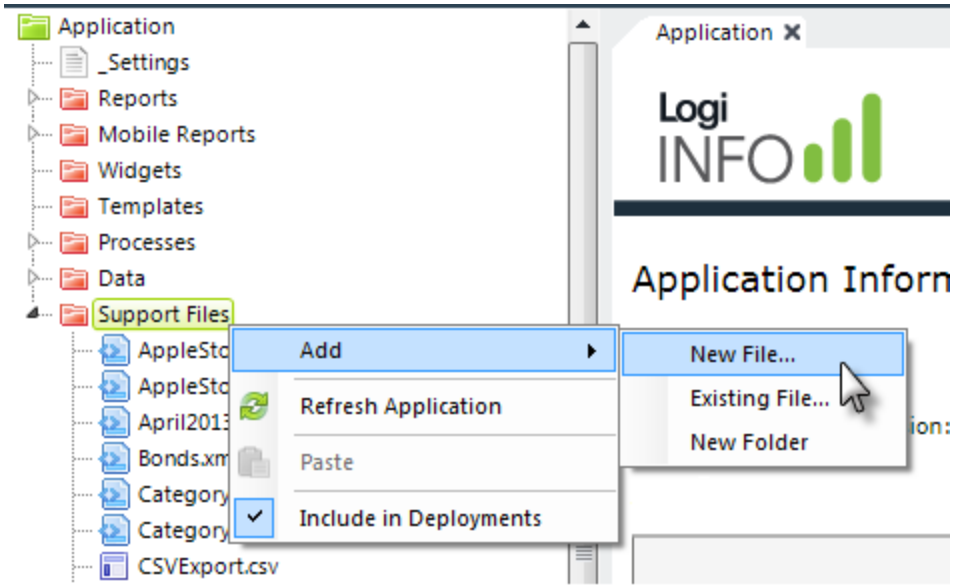
If you're wondering what those "TH" and "TD" designations are, they relate to the tags that make up an HTML table: TH = Table Header, and TD = Table Data. By including them as shown above, their classes will be automatically applied to any Data Table that uses the myTable class. This is a bit of CSS shorthand that saves you time and effort.

Including a Style Sheet in Your Application

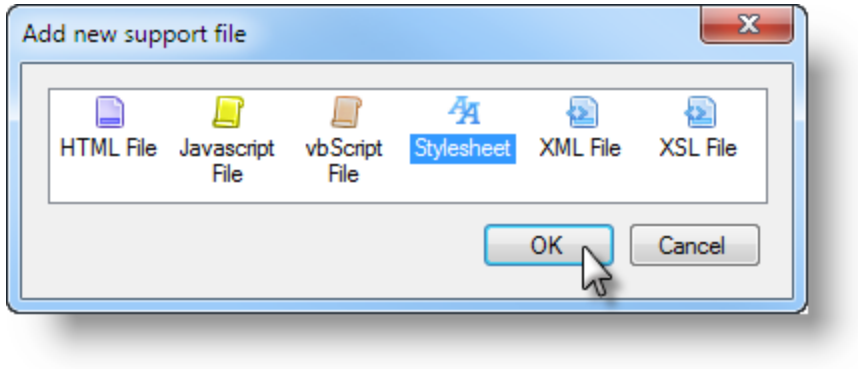
Logi Info is CSS-compliant and Logi Studio includes a style sheet editor to ease development. From the Logi application perspective, a style sheet is a *support file* that's included with a report application. In Logi Studio's Application panel, the Support Files folder provides an access to application style sheets. Style sheets can be used in conjunction with built-in or custom Themes.

This topic uses screen shot images based on a fictional Logi application that pulls data from the Northwind Foods database. If you already have a basic Logi database table report created, you can use it as we proceed.

Creating a New Style Sheet File



1. In Studio, as shown above, right-click the **Support Files** folder in the Application Panel.
2. Click **Add**, then **New File...**



3. A dialog box will be displayed that allows you to choose the type of support file you wish to add; select **Stylesheet** and click **OK**.
4. A new file, named *newStyleSheet.css* will be added to the list of support files, and opened for editing in the Workspace Panel. An empty BODY class will be inserted by default:

```
BODY
   {


}
```

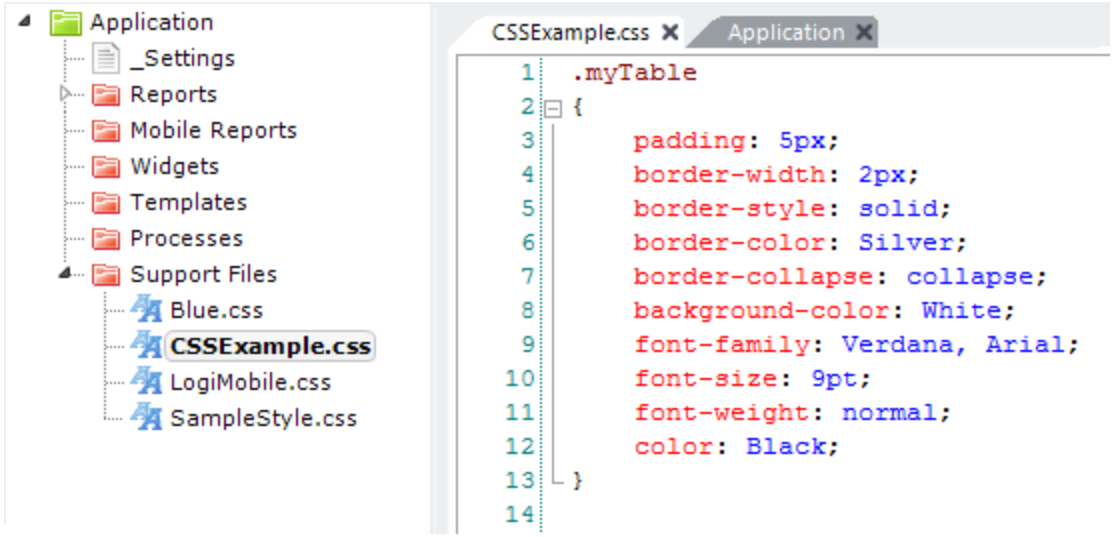
5. **Rename** the new style sheet by selecting it in the Application Panel and pressing **F2**, or by right-clicking it and selecting Rename from the popup menu.

Add an Existing Style Sheet File

1. For an existing file, select **Existing File...** from the popup menu shown earlier and use the file browser to navigate to the directory where the style sheet is located. Select the style sheet file and click **Open**.
2. The file will be *copied* to the _SupportFiles folder in the application project folder and added to the Support Files list in Studio.

Editing a Style Sheet in your Studio Workspace

There are several ways to edit style sheets from Logi Studio and the most direct way is to do it right in the Studio Workspace editor:



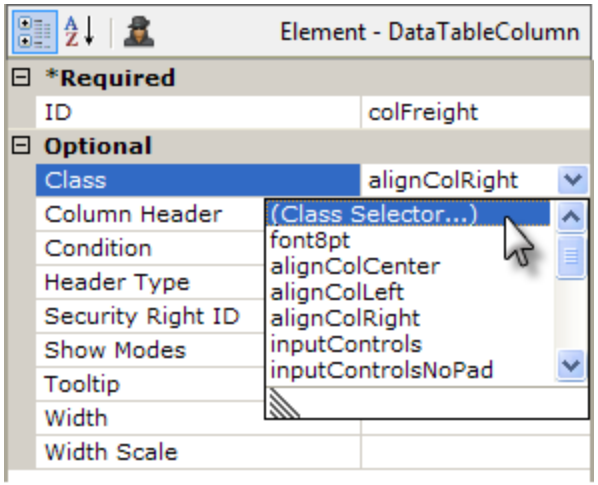
1. In the Application panel's **Support Files** folder, double-click the style sheet file to be edited.
2. The style sheet contents are displayed in the Workspace panel, as shown above, and can be directly edited.
3. As you type, the editor will provide **assistance** via pop-up lists of selector names and values.
4. Make the necessary changes or additions and click **Save** on the toolbar.

Use of the Workspace **Preview** tab automatically saves definition files but *does not* save changes to Support files, such as style sheets. You must click Save to save your changes before previewing.

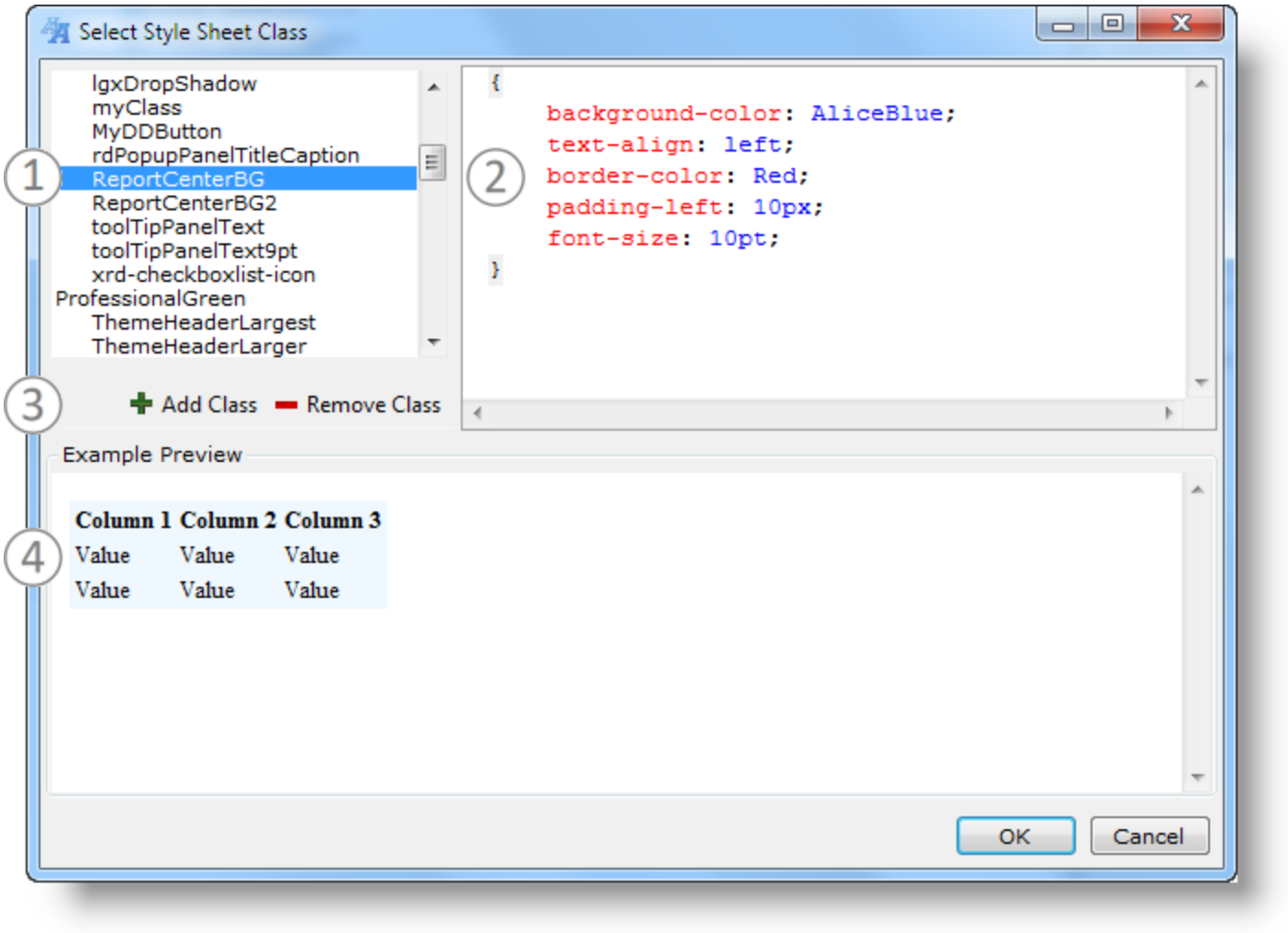
You can also select and right-click a style sheet file in the Application panel, and click **Open Externally** to edit it in any external CSS editing tool that you've installed. The file will be opened in whatever application is associated with the .css file extension.

Editing a Style Sheet in the Style Sheet Class Selector

All Logi Info elements have attributes associated with them and they can been in Logi Studio in the Attributes panel when an element is selected. Many elements have a **Class** attribute and its value field includes a drop-down list icon. Clicking this icon causes a list of classes from the style sheet(s) assigned to your application (described in "Editing a Style Sheet Using the External Style Sheet Editor" on page 194) to be displayed. The top item in this list is a link to the *Class Selector* tool, which offers a second method of previewing and editing your style classes.



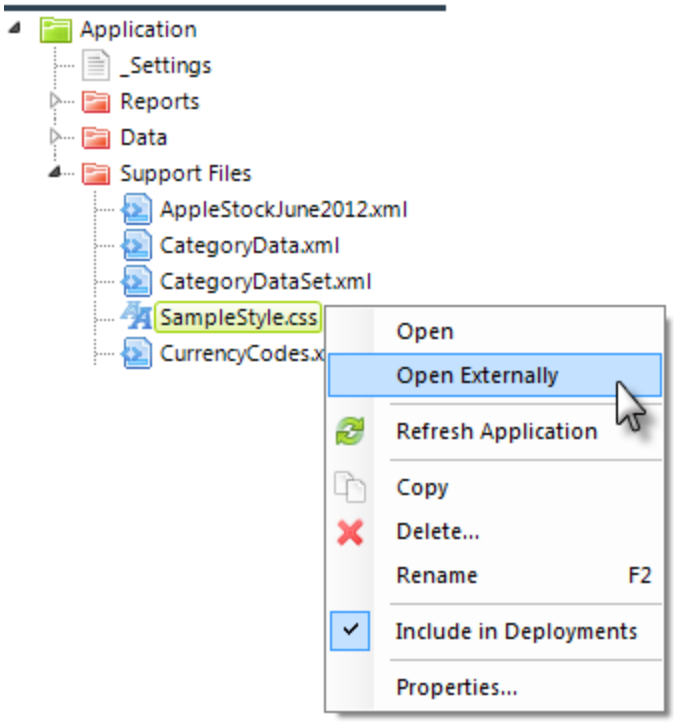
The Class Selector tool can be opened by selecting it from the list ofClass attribute value choices for any element, as shown above.

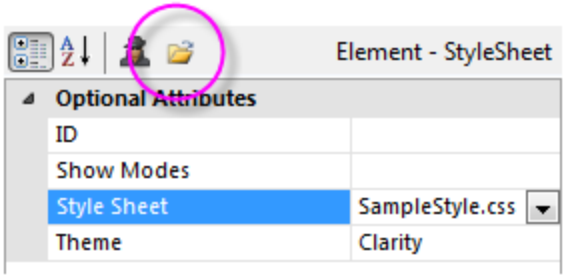


The Class Selector tool, shown above, allows a class to be assigned by selecting it in the class list (1) and clicking **OK**. The attributes of the selected class are shown in the upper right panel (2) and they can be edited there. Controls (3) allow you to save, add, and delete classes, and the effect of the selected class is shown in the lower Example Preview panel (4).

Editing a Style Sheet Using the External Style Sheet Editor

The external editing option works only if you have installed a 3rd-party style sheet editor program, which has been associated in the file system with the .css file extension. This external editor can then be launched from within Studio in two different ways:



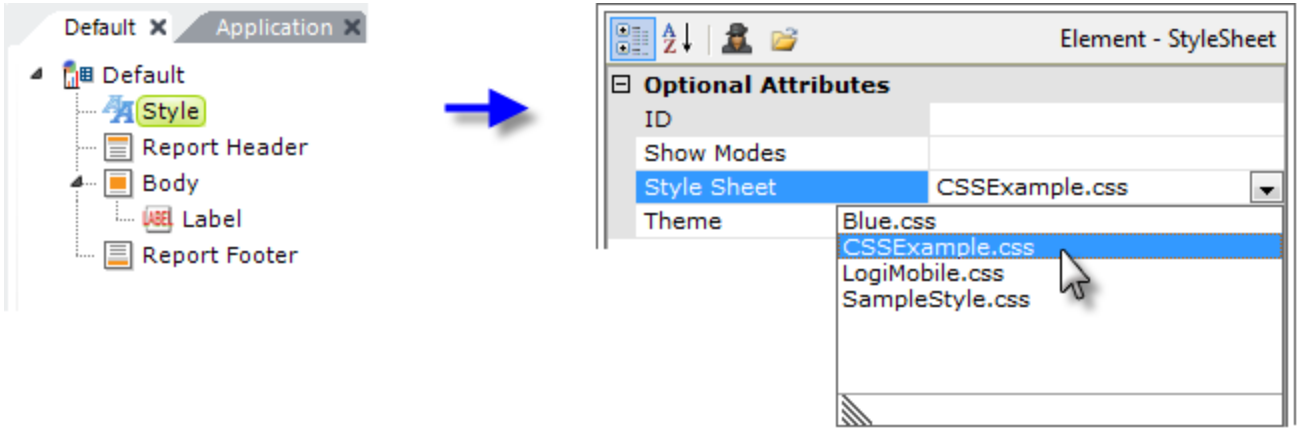


A style sheet file can be selected in the list of files in the Application panel, right-clicked, and opened externally, as shown above left. Or in a report definition, when the **Style** element's **Style Sheet** attribute has been selected in the Attributes panel, as shown above right, the **Open Referenced File...** icon will appear and can be used to open the file in the external editor.

If you've edited a style sheet in an external editor, you may need to "refresh" the file in Studio before any changes are recognized. This is done by right-clicking the file in the Application panel and selecting **Refresh Application**.

Assigning a Style Sheet to Your Definition

Once the CSSExample.css style sheet file has been copied to your Logi application folder, you have to assign it to your report definition.



1. Open your report definition in the Workspace panel and select the **Style** element, as shown above.
2. In the Attributes panel, click the **Style Sheet** attribute and pull-down the list of available style sheets.
3. Select the *CSSExample.css* style sheet entry in the list. Now the classes in that style sheet are available for use in your definition.
4. You can open the selected style sheet in an external editor, if you have installed one, by clicking the **Open File** icon at the top of the panel. This will launch the application associated in the file system with the .css file extension and open the file in it.

Assignment Using a URL

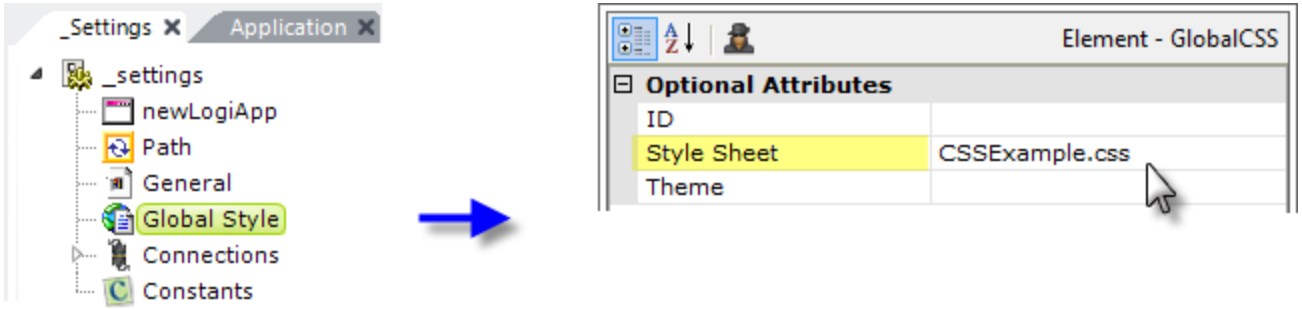
If a style sheet resides outside of your Logi application, it can instead be assigned using a fully-formed URL, such as

```
https://www.myFirm.com/CSS/myStyleSheet.css
```

Just enter the URL into the Style Sheet attribute, instead of selecting a file name from the drop-down list.

Global vs. *Local* Style Sheet

The presence of the Style element in a report definition indicates use of a *local* style sheet. A Logi application usually contains many report definitions and each definition can use a separate style sheet but, more often, it's convenient to use a *global* style sheet. A global style sheet is configured in the _Settings definition:



1. Open the _Settings definition and add a **Global Style** element to it, as shown above.
2. Select the newly added Global Style element in the Workspace panel.
3. In the Attributes panel, enter a value for the Style Sheet attribute or select one from the pull-down list.

Assigning Classes to Elements

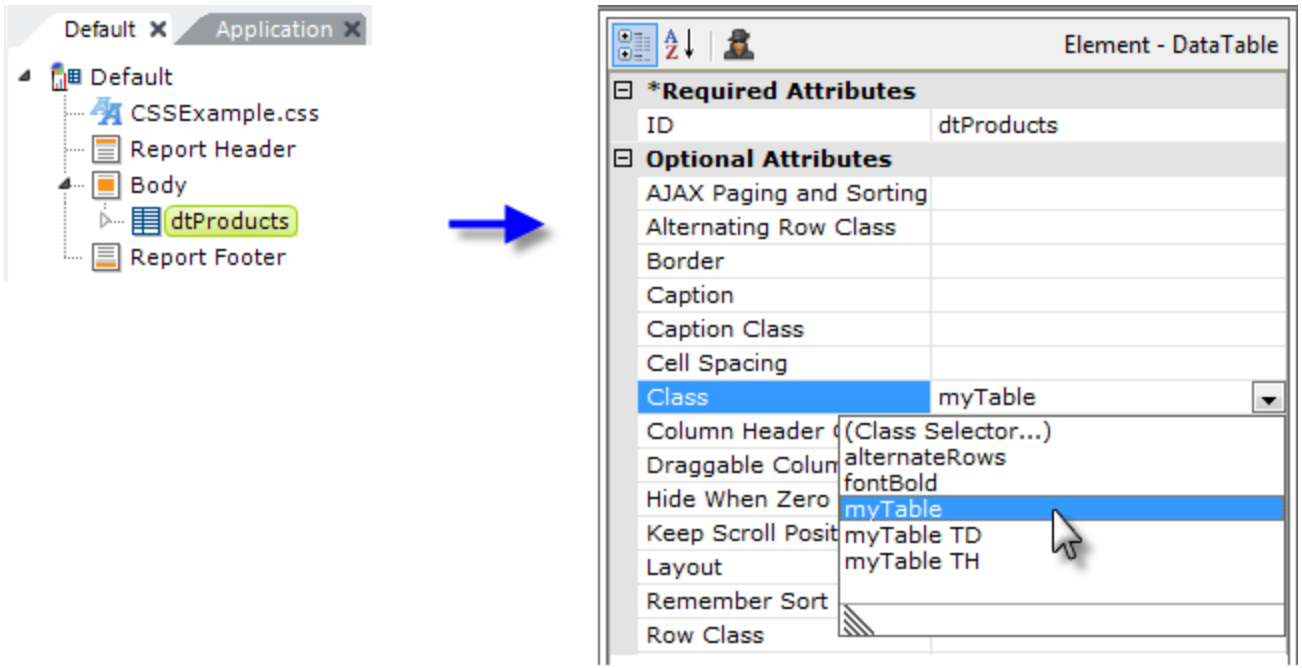
You can assign style classes from the style sheet to elements in your application. The following example assumes this query:

```
SELECT CategoryID, CategoryName, Description FROM Categories
```

has been used to produce a report from the Northwind Foods database. Follow along in your application, making adjustments as necessary. These examples assume you've already assigned the CSSExample.css style sheet to your application.

CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

The report output above shows the data returned by the query. It's raw and not very attractive, so let's assign some style classes to improve its appearance. Elements that can be directly affected by style have a **Class** attribute. Some may have several class-related attributes depending on the complexity of the element.



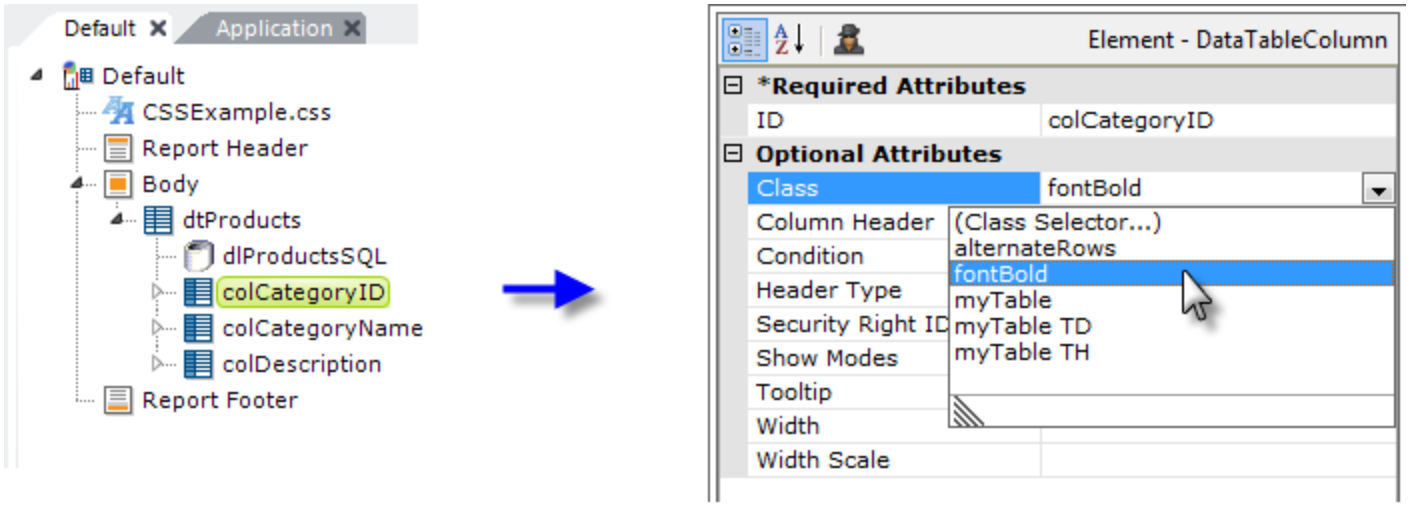
In your report definition, select the **Data Table** element and, in the Attributes panel, find its Class attribute and, using the drop-down list, select the *myTable* class, as shown above.

The list includes *all* of the classes in your style sheet. It's also possible to manually assign *multiple* classes, separated by commas or spaces, as a Class attribute value.

Now let's preview the report:

CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

The table, with the style sheet classes applied, is shown above. It looks much better than the raw table! The style sheet has affected font sizes and colors, text alignment, background color, spacing and padding, and borders. The classes affect everything about the table and its contents, flowing down through the Data Table element and all its child elements. This is the "cascading" effect of Cascading Style Sheets. However, the effects can be overridden by assigning other classes to child elements lower down in the hierarchy. Let's see how that works. With the myTable class assigned to the table, all the data in all the columns appears in a normal, black font. Let's change the data in the first column, the CategoryID, to be bold-faced as well.



As shown above, select the **Data Table Column** element for the Category ID and, in its Class attribute, select the *fontBold* class from the drop-down list.

CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

As you can see above, the class applied to the data column has produced an additive effect, in that column only. We could just as well have assigned the class to the Label element that actually displays the data. By assigning it to the Data Table Column, we applied the affect to all its child elements, which covers situations where you have multiple Labels or other elements in the column.

Create an Alternate Row Class

If you're using a Logi stock theme, it comes with a *ThemeAlternatingRow* class that you can assign to your Data Table element's **Alternating Row Class** attribute. If you're not using a theme, or want to create your own class for alternating Data Table rows, the following exercise shows you how to do that. Add a new class to the style sheet and then assign it to the Data Table. The goal is to have every other row in our table look slightly different, which makes it easier to read across many columns.

1. In Studio's Application panel, double-click and open the CSSExample.css style sheet file.
2. In the Workspace editor, add the following CSS class:

```
.alternateRows {
background-color: LightYellow;
}
```

3. Click the **Save** icon in the toolbar, to save the style sheet changes.
4. Click the tab at the top of the Workspace panel to return to your report definition.
5. Select the Data Table element and assign your new *alternateRows* class to the **Alternating Row Class** attribute.
6. Preview the results:

CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

Create Dynamic Row Highlighting

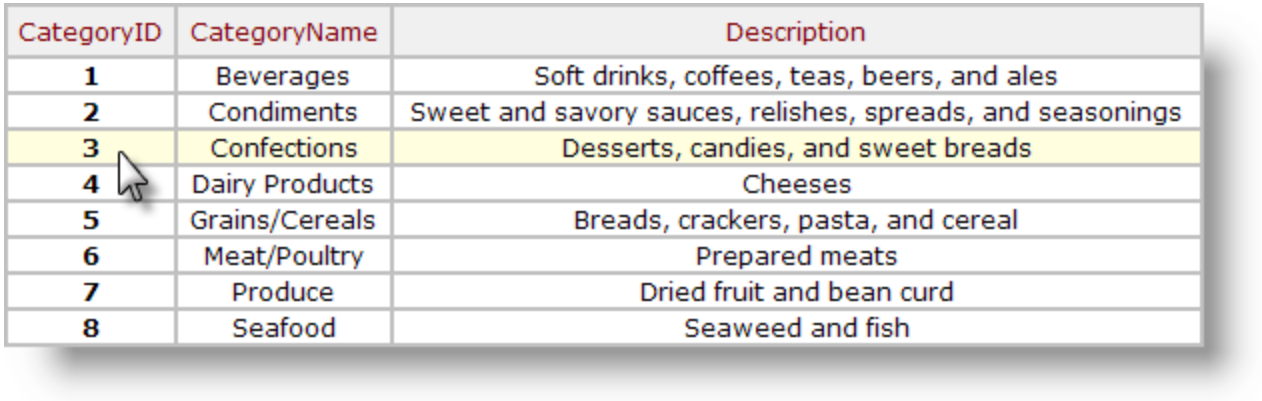
If you're using a stock Logi theme, it automatically applies dynamic row highlighting. However, if you're not using a theme, or want to create your own class for dynamic row highlighting, the following exercise shows you how to do it.

This effect may not work with all browsers (especially older browsers):

1. Once again, open your CSSExample.css style sheet file.
2. In the Workspace editor, add the following CSS class:

```
.myTable TR:hover TD {
background-color: LightYellow;
}
```

3. Click the **Save** icon in the toolbar, to save the style sheet changes.
4. Click the tab at the top of the Workspace panel to return to your report definition.
5. Select the Data Table element and *delete* the **Alternating Row Class** attribute value, leaving it blank.
6. Preview the results, and pass your mouse cursor over the rows:



CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

As you have seen, style sheets are **powerful tools** that assist you in creating visually-interesting report presentations. More information about style sheets and Logi applications can be found in and "Using Style Sheets" on the next page.

Using Style Sheets

Cascading Style Sheets (CSS) are a powerful technology that enable you to control the appearance of your Logi applications.

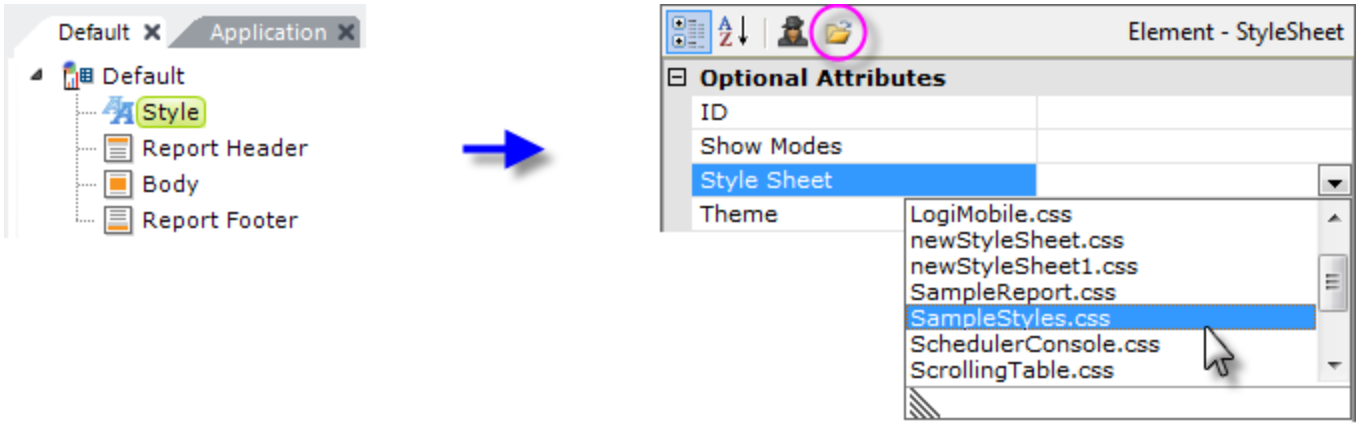
The following topics introduce the process of applying a style sheet to various parts of a report definition and style classes to elements:

- Applying a Style Sheet
- Assigning a Style Class to an Element
- Using Conditional Class Elements
- *Global*versus *Local* Style Sheets
- Supported Class Definitions
- Overriding Element Style Classes
- Position May Matter
- Prohibited Characters

New to CSS? See "Style Sheets" on page 181 before proceeding.

Applying a Style Sheet

Once a style sheet file has been added to an application's _SupportFiles folder, it's available for use with any of the definitions within that application.



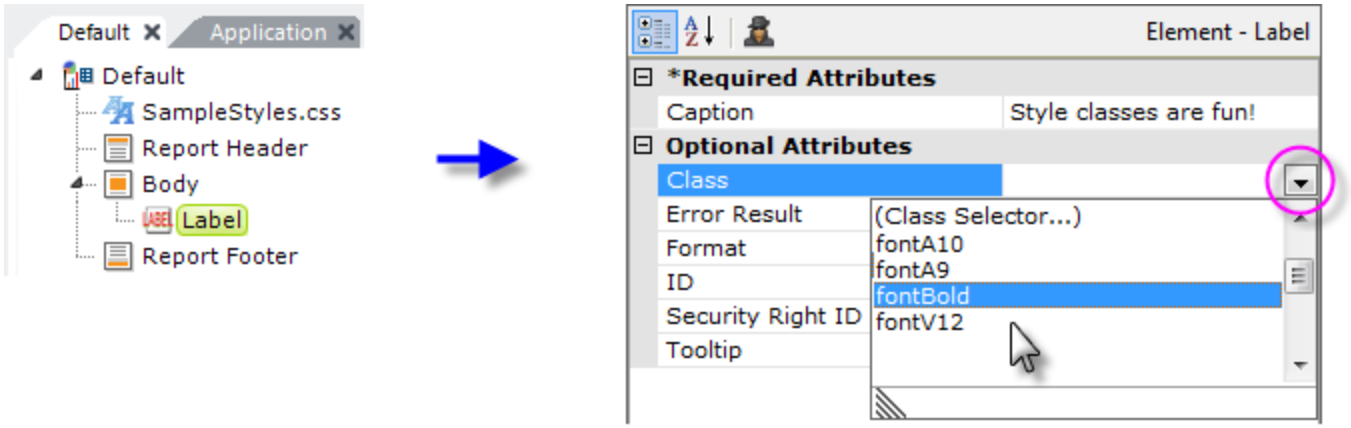
1. To apply a style sheet to a specific definition, open the definition in the Workspace, as shown above.
2. Add a **Style** element, if necessary,
3. The Style element's **Style Sheet** attribute will provide a pull-down list of the style sheet files in the _SupportFiles folder. Select the appropriate style sheet for use with this definition.

Once a style sheet has been selected, it can be edited in an external style sheet editor (if you have installed one) by clicking the Open File... link (shown circled above) at the top of the Attributes panel. This will launch the application associated in the file system with the .css file extension and open the file in it.

You can assign multiple style sheets but adding multiple Style elements in your definition. If two style sheets contain a class with the same name, then the last one (identified in the Style element further down the element tree) will "win" any conflict, overriding earlier classes.

Assigning a Style Class to an Element

Once a style sheet has been applied to the report, the next step is assigning style *classes* to individual elements. Class assignment provides flexibility for the layout and appearance of a report definition.



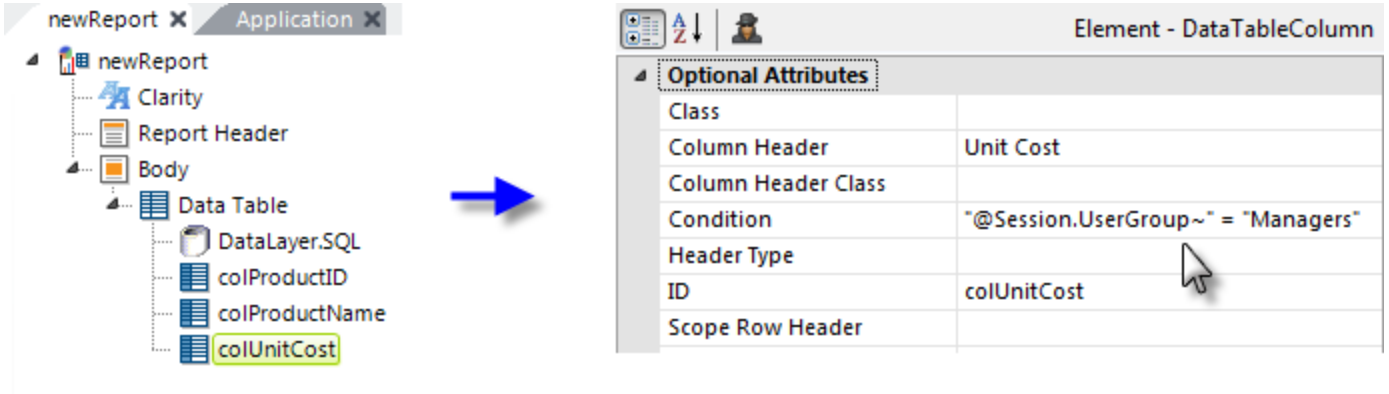
1. To apply a style class to an element, select the element in the Workspace panel in order to view its attributes.
2. In the Attribute panel, find the element's **Class** attribute and use the drop-down list (click the down arrow) to select a class from those available in the style sheet assigned to this definition, as shown above. If no classes appear, then no style sheet has been properly assigned.

   Alternately, you can type in a style class name, or select it by invoking the **Class Selector** tool, which is the first entry in the list. You can also assign *multiple* style classes to an element by entering their names in the element's Class attribute, separated by a space or a comma.

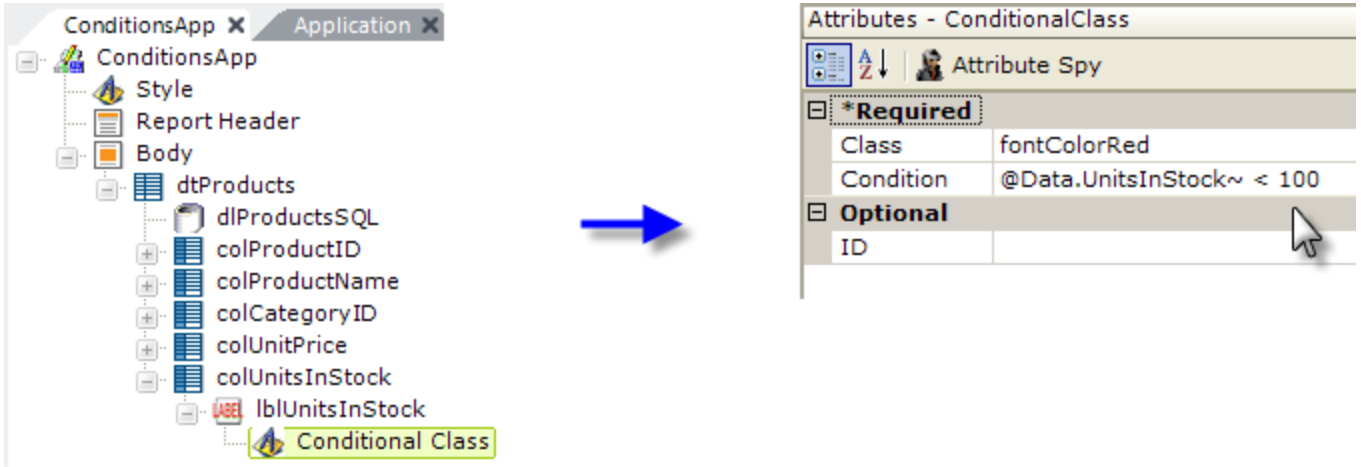
Sample applications installed with Logi managed reporting products include sample style sheets, which you can play with and modify.

Using Conditional Class Elements

The **Conditional Class** element allows you to dynamically apply different style sheet classes, based on the Condition attribute.



Suppose product managers want to be *visually alerted* if product stock levels fall below 100 units. In fact, when an item falls below that threshold, they'd like to see its "in stock" count displayed in the table in red, as in the example above.



In the example shown above, the **Label** in the last Data Table column displays the number of units in stock. Its font color may be explicitly set in its own Class attribute, or it may be inherited from a parent element or a theme.

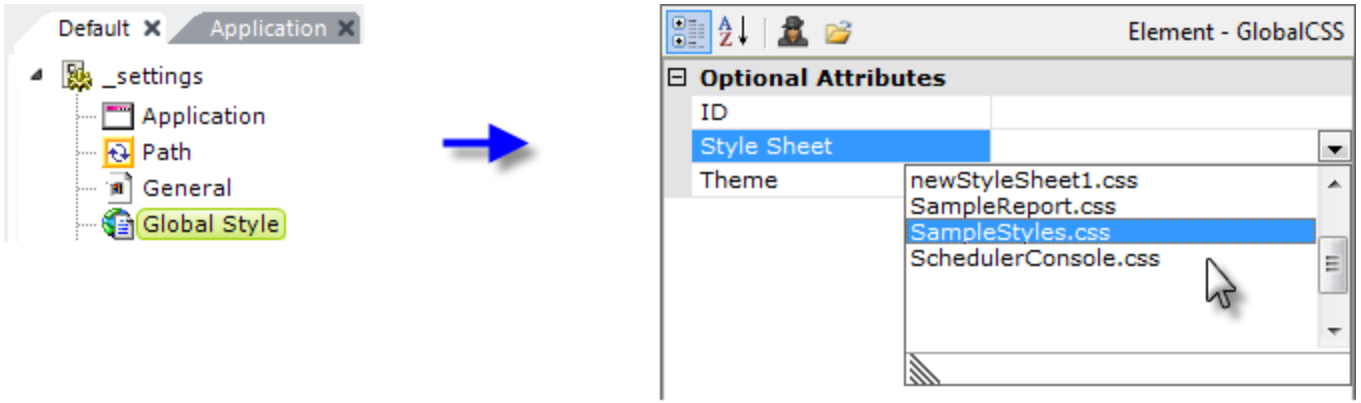
To give the product managers what they want, a **Conditional Class** element is added beneath the Label, and its attributes set as shown above. The expression in the Conditional Class element's **Condition** attribute sets the threshold and its **Class** attribute specifies which class to apply if the Condition expression evaluates to *True*.

Multiple Conditional Class elements can be used beneath a parent element. In this case, the class from the *first* one of these elements that has a Condition attribute that evaluates to *True* will be applied; however, any remaining Conditional Class elements below it will *not* be evaluated.

 If you're using a stock Logi Theme (see "Themes" on page 235) and use a Conditional Class element with a complex element structure, like a Data Table, application of the conditional class may override the Theme styling for the *entire* structure, causing undesired results. Be sure, especially when using a Theme, to apply conditional classes at the right level of the element structure. For example, if you want to affect Data Table rows, make a Conditional Class element the child of each Data Table Column element, rather than making it a child of the Data Table element itself.

Global versus Local Style Sheets

The presence of the **Style** element in a report definition indicates a *local* style sheet, one that applies only to that definition. A Logi application may contain several report definitions, and each definition may use a separate style sheet. However, a *global* style sheet can be used instead, one that affects *all* report definitions in an application. The global style sheet is configured in the _Settings definition.



1. Double-click the _Settings definition in the Application Panel, so that it opens for editing in the Workspace panel.
2. In the Element Toolbox panel, double-click the **Global Style** element to add it to the definition.
3. Select the Global Style element, as shown above, and select a style sheet from the pull-down list in its **Style Sheet** attribute.

Now all of the applications definitions will be able to use classes from the global style sheet and you need not assign a style sheet to each definition individually.

However, what happens if you assign *both* a local *and* a global style sheet, and each has a class with the same name but with a different class definition? As you might guess, the local style sheet's classes will override those of the same name in the global style sheet. This is discussed in more detail in "Position May Matter" on page 217.

Supported Class Definitions

There are several ways in which classes can be defined within style sheets and all of them can be used in style sheets for Logi applications.

Logi elements work with CSS classes that are defined using these selectors:

Selector	Example	How Applied

1. An HTML tag	BODY {
	color: white;
	}
		Applied automatically to the HTML tag. No assignment in an element's Class attribute is necessary.

2. An HTML tag and a Class name, separated by a period (.)	TD.left {
	text-align: left;
	}
		Applied automatically to the HTML tag. For example, if a table element has the class *.left* applied to it, its cells (which use <TD> tags) will have *TD.left* applied to them.

3. A Class name only, preceded by a period (.)	`.textLeft {`
	`text-align: left;`
	`}`
		Applied to an element by entering the class name, *without* the leading period, in the element's Class attribute.

4. An Element ID, preceded by the hash mark (#)	`#myLabel {`
	`color:Red;`
	`}`
		Automatically applied to elements with an element ID matching the class name. It's not available to any element with a different ID. No assignment in the Class attribute is necessary.

The last example, which uses the Element ID, is very useful for applying style to elements which seem to have no mechanism for controlling their appearance. For example, by default, Data Table Column header text is centered and its data is left-aligned, but this technique can be used to right-align them instead.

Overriding Element Style Classes

The acronym "CSS" stands for Cascading Style Sheet, which means that the effects of style sheets, like water, flow "down hill". Classes assigned to elements that are "containers" or parents for other elements, such as divisions, tables, or rows, affect all the elements they contain. The style "flows" down into all the child elements within the container. However, if a style class is individually assigned to one of the child elements, it can *override* the style set in the parent container element.

For example, you could assign to a **Table** element a style class that includes `text-align: left`, causing all text in the table columns to be aligned to the left. However, for a numeric column, you could assign a class that includes `text-align: right` to one of the **Data Table Column** elements (a child element of the Table element). The alignment specified by the Table element's class will be overridden by the Data Table Column element's class.

Similarly, classes from a global style sheet are applied before classes from a local style sheet, so the local style sheet will be the winner (will override) if there are any class duplications in both style sheets.

This inheritance and overriding of style classes doesn't always work the way you expect it to, though, and it can be frustrating. If style assigned to a parent isn't working on the child elements as you expect, ensure that they don't have their own class assignments or try moving the class assignment "up" a level, to the parent container element's parent container.

Some complicated elements, such as the **Dashboard**, **Analysis Grid**, and **Tabs** elements, have their own "internal style sheets" which govern their general appearance. It is possible to affect their appearance by including appropriate classes in your own report-level style sheet that override the default classes used with the elements. By looking around in your application folder, under the rdTemplate folder, you can find these element style sheets and determine which classes to include and thereby override in your own style sheet.

 *Never change the default style sheet distributed with your Logi product! Always work from a copy.*

There's a caveat, however: future releases of Logi products may include class name changes that could cause your overrides to stop working and require you to update them.

Style Sheets vs Themes

Logi **Themes** apply a specific appearance to a report page. A Theme includes its own style sheet and it's possible to assign *both* a theme *and* an individual style sheet to a report definition. In this case, if the theme's style sheet conflicts with your individual style sheet, then the classes in your individual style sheet will "win".

How Did They Do That?

Not sure how to *recreate* the styles you see on the web? There are developer tools built into most modern browsers that let you to look "into" web pages to see how style is applied in them. These tools can be very helpful in learning both commonly-used and more esoteric techniques, and also in understanding how classes are "cascading" through the web page. They're are usually invoked using the F12 key while viewing a page in the browser.

In addition, unless a style sheet has been "minified" or compacted to an almost unreadable state, you can view it to see its classes. This is a great way to learn CSS tricks and advanced techniques. Look for its URL near the top of an HTML page, copy it, and paste it into your browser's address bar (relative references may require a little extra work), then browse to see the classes.

Position May Matter

Because of the way that the HTML is generated when a Logi application is run, the *position* of a local **Style** element in the Element Tree may be significant. It may affect its ability to override classes in other style sheets, especially for individual elements such as Dashboards and Analysis Grids.

Due to the variety involved in this, it's difficult to provide definite rules here. However, if you're attempting to override other classes and find it's not working, and have checked all other variables such as correct spelling, etc., try repositioning the Style element *lower down* in the element tree so that it's below the element you're trying to affect.

Prohibited Characters

As a general rule, you should avoid using any **un-encoded, invalid XML** characters in your style sheets... even in comments. For example, consider this style class:

```
BODY {
margin: 0px auto;
background-color: #F6F6F6;
font-family: Verdana, Arial, Helvetica;
text-align: -moz-center; /* centering for Firefox */
text-align: -khtml-center; /* centering for Safari &Chrome */
#text-align: center; /* centering for IE */
}
```

When used with report definitions that are manipulated by elements, such as Procedure.Send Html Report, that stream the CSS code into the report XML, the class shown above will cause an error, because there's an un-encoded ampersand (&) in the high-lighted comment.

The best practice is to simply avoid these characters - in the comment above, use of the word "and" would do the trick. You could provide them in encoded form (for example, "&" for ampersand) but in something like a comment, it's less trouble and easier to understand if you just spell it out.

Working with Font Awesome

Font Awesome is a free library of more than 600 scalable vector icons that can be sized, colored, and arranged using CSS.

The following topics explain how to use it in your Logi application:

About Font Awesome

Font Awesome is a free library of more than 600 scalable vector icons that can be sized, colored, and arranged using CSS. It doesn't require scripting, works well with a variety of devices, and is easy to implement in Logi Info apps. Here are some examples:

- Bluetooth
- Gauge
- Line Chart

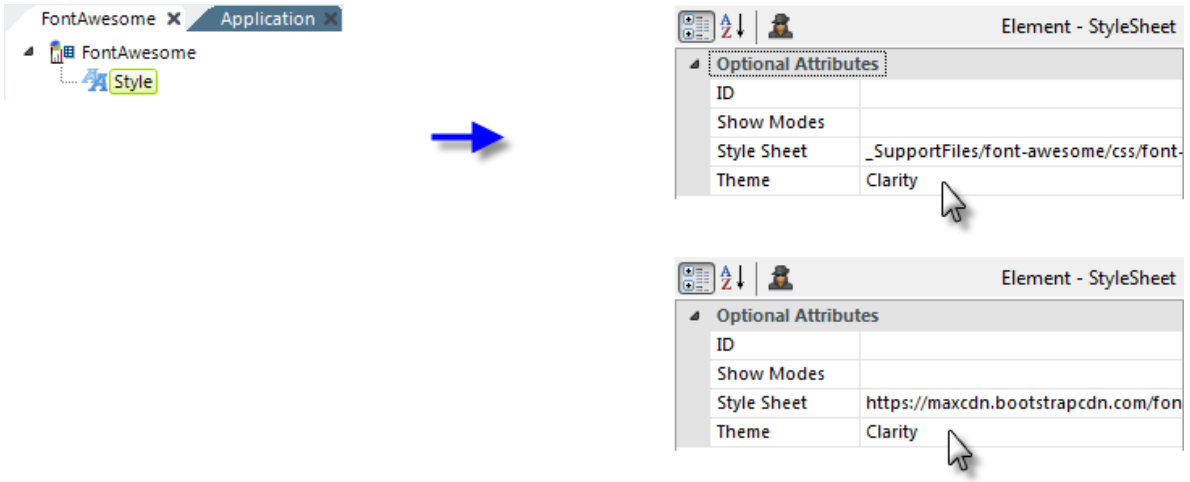
- Navicon
- Arrow Up

Here's a link to the official Font Awesome web site. This topic explains how to use icons from this great collection in a Logi application. You can view and save a Logi Info definition that includes all of the examples in this topic here. Just like a Logi Sample App page definition, be sure to open it in your browser, right-click it, and select "View source" before copying any parts of it to Logi Studio.

 You cannot successfully email a report containing Font Awesome icons using the **Action.Email Report** or **Procedure.Send HTML Report** elements.

Including the Font Awesome Library

First, you need to include the Font Awesome (FA) library. You can either download it from the FA web site and store it in a sub-folder in your Logi app's _SupportFiles folder, or link to it directly.



Examples of both approaches are shown above. For a local copy of the library, use this path to the FA style sheet:

```
_SupportFiles/font-awesome-4.5.0/css/font-awesome.min.css
```

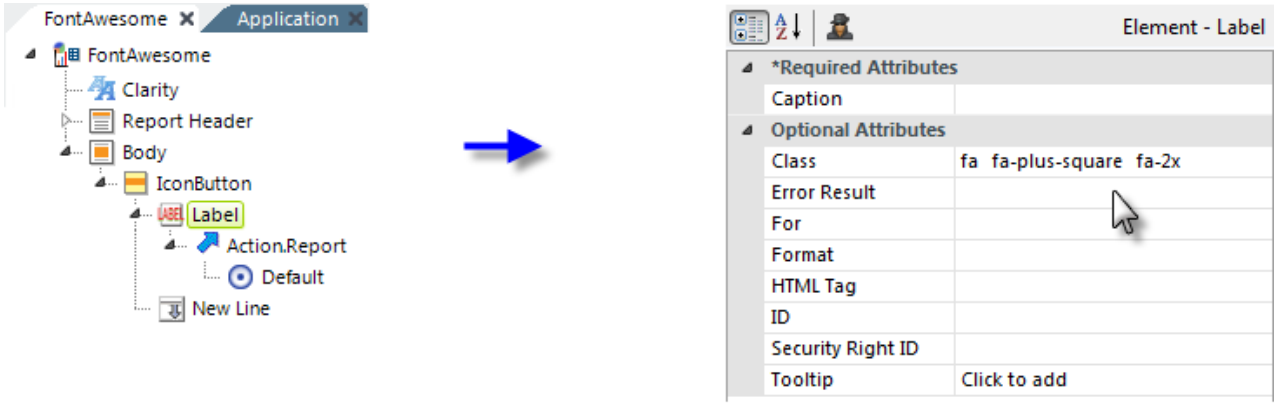
To use the online version instead, use this URL:

```
https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css
```

You can also use any of the standard Logi built-in Themes with the FA library.

Adding a Simple Icon

One of the simplest ways to add an icon to a report is to use a **Label** element.



The basic procedure is to go to the FA Icons web page where all the available icons are displayed, find the icon you want to use, and click it. You'll be taken to a detail page for that icon and you'll see the HTML for implementing it, which looks similar to this:

```
<i class="fa fa-plus-square"></i>
```

Select and copy the class attribute from that code (highlighted in yellow) and paste it into your definition, in the Label element's **Class** attribute value, as shown above.

Class names are *case-sensitive*! In order to size the icon (relative to its container), you can use one of these additional classes, in combination with the icon class:

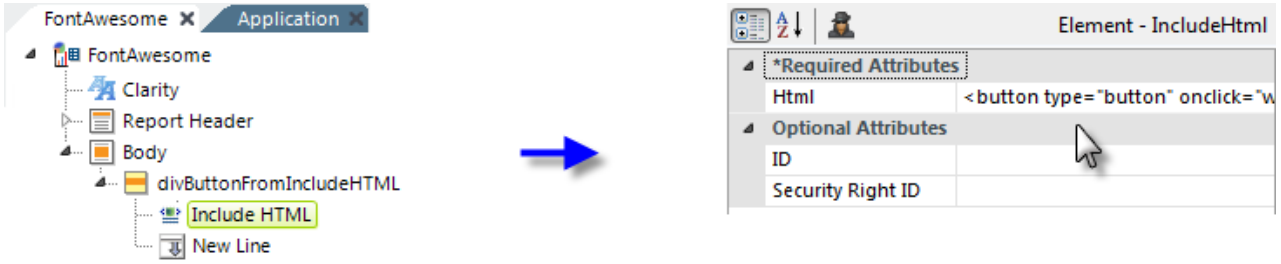
```
fa-lg, fa-2x, fa-3x, fa-4x, fa-5x
```

Classes for sizing and other usage details are discussed in the FA Examples web page. We've exaggerated the spacing in the Class attribute value in the image above to make things a little clearer. One space between classes is all you need. And the resulting icon looks like this:

Give it a try and experiment with different sizes.

Adding a Button using HTML

Of course, if you want to, or need to, add icons using the actual HTML, it can be done by using the **Include HTML** element.



In the example shown above, an Include HTML element has been added and its HTML code is set to:

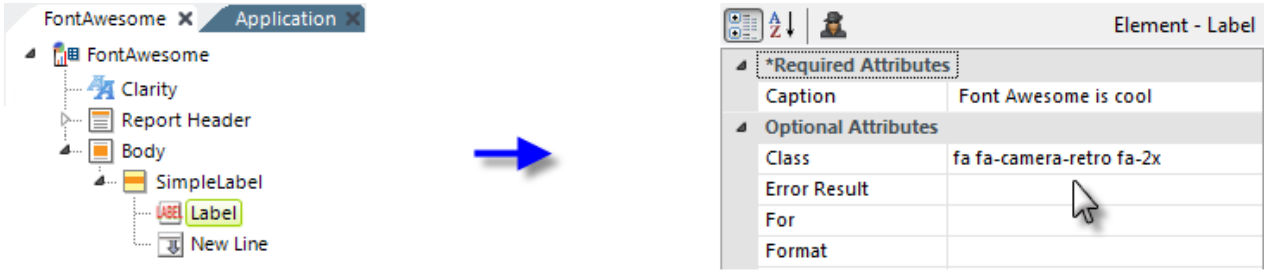
```
<button type="button" onclick="window.location.href='http://www.google.com'">
 <i class="fa fa-chevron-circle-up fa-2x"></i>
</button>
```

which uses the *onClick* event to trigger an action when the button icon is clicked.

And the resulting icon is shown above.

Combining Icon and Text in a Label

You can combine an icon with the caption of a **Label** element:



As shown above, a Label element is given a **Caption** attribute value and set to the FA class for the desired icon.

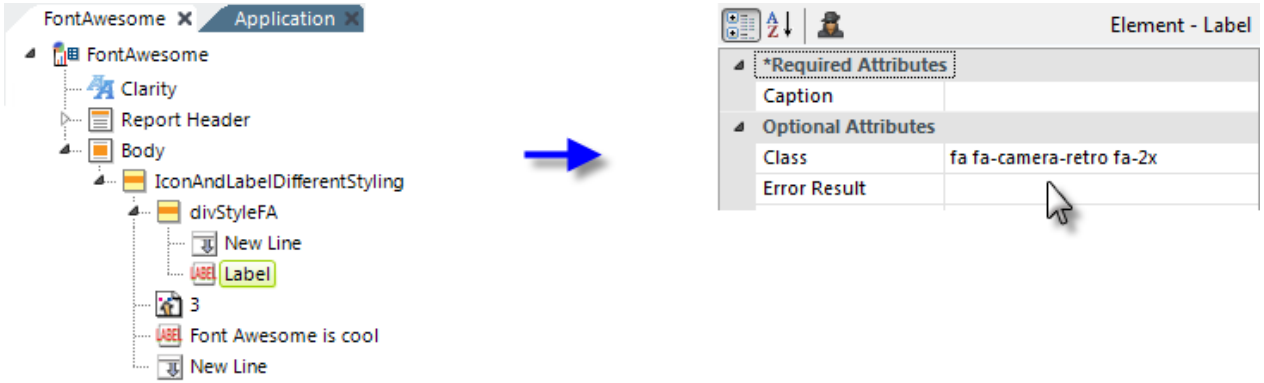
Font Awesome is cool

And the result is shown above.

- If you want to add more space between the icon and the text, add a space at the beginning of your Caption attribute (only *one* - any more than that will be ignored by the browser).
- The style of the caption *text* is controlled by the FA classes. You can't change it by adding more classes to the Class attrib-

ute.

However, if you want to style the icon and the text separately, you can do this:



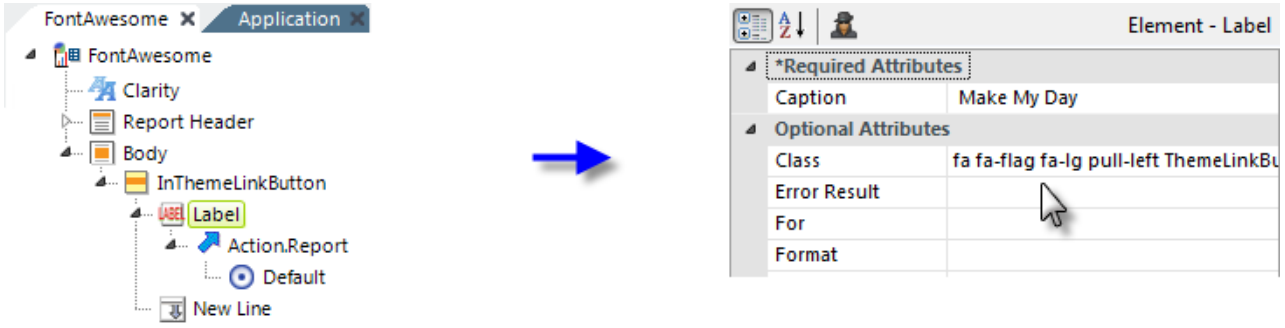
In this example, separate Label elements are used for the icon and the accompanying text. The Label element for the icon is also placed in a separate Division, which allows the icon to be further styled, in this case by setting its Color using CSS in a separate style sheet.

Font Awesome is cool

By applying style to the Division element ("divStyleFA") that contains the icon, the icon color has been set to *Blue.* The second Label element's Class attribute has been set to change the font size of the text.

Using an Icon with a Theme Link Button

In "Combining Icon and Text in a Label" on page 225, when an icon was used in a Label with a caption, we couldn't add more classes to style the text. That remains true but we *can* add other classes to style the text *container*. In this example, we'll add the ThemeLinkButton class.



As shown above, we've used a Label element, with a caption, and we've set the Class attribute value to:

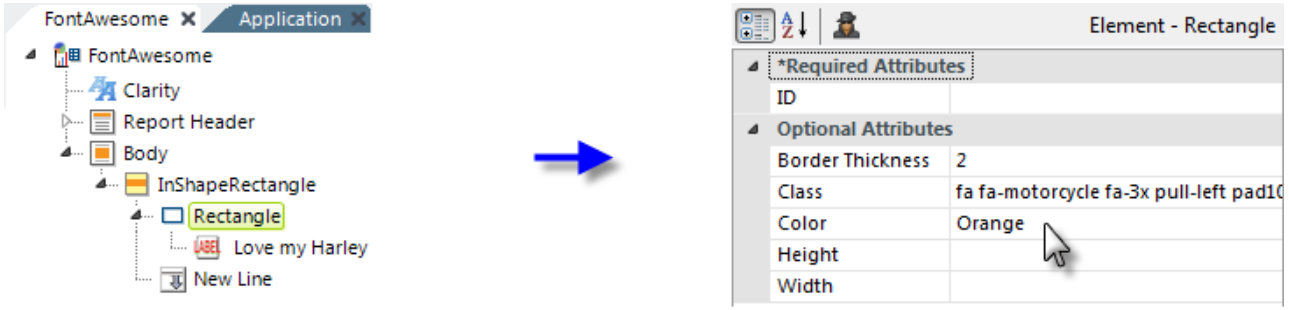
```
fa fa-flag fa-lg pull-left ThemeLinkButton
```

The *pull-left* class is an FA class that left-justifies the icon and the *ThemeLinkButton* class has also been added.

And we can see the resulting Theme-based button shape and coloring above. In a live application, the button shading will change when the mouse cursor hovers over it.

Using an Icon with a Shape Element

Logi Info includes a few standard shape elements, such as **Rectangle**. Let's see how we can use an FA icon with that.



In the example above, we see a Rectangle element with a child **Label** element (the Label text will appear inside the Rectangle borders). The Rectangle's **Class** attribute is set to:

```
fa fa-motorcycle fa-3x pull-left pad10
```

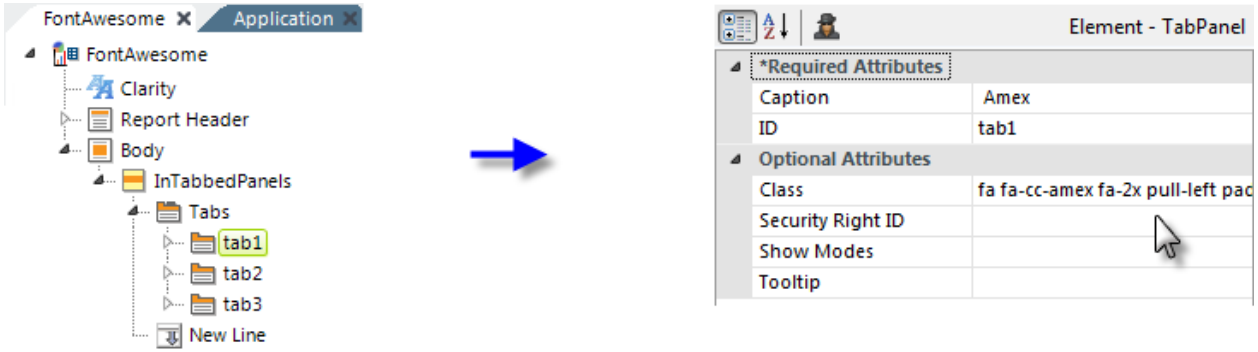
The only class you won't recognize is *pad10*, which is in a separate style sheet added for this example app and provides some padding inside the rectangle borders.

And the results are shown above.

The height of the rectangle and the size of the text is being controlled by the *height* of the icon + the padding.

Using Icons in Tabbed Panels

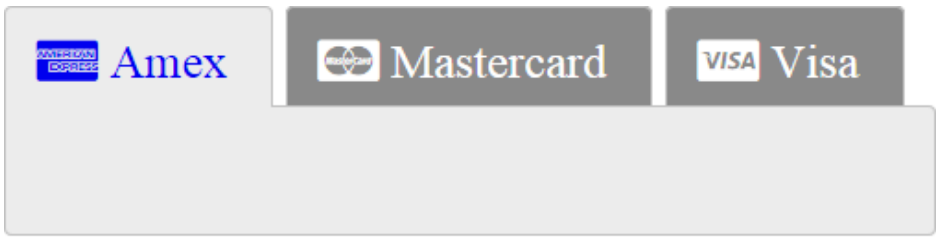
Finally, let's see how we can use multiple icons in tabbed panels.



In the example above, a **Tab** element and several **Tab Panel** elements have been added. Each panel is supposed to contain information about different credit cards, so let's place their names and an icon for each card in their respective tabs. We can do this with a Class attribute value of:

```
fa fa-cc-amex fa-2x pull-left padTB10
```

The *fa-cc-xxxx* class will be different for each type of credit card. The only class you won't recognize is *padTB10*, which is in a separate style sheet added for this example app and provides some top and bottom padding inside the tabs.

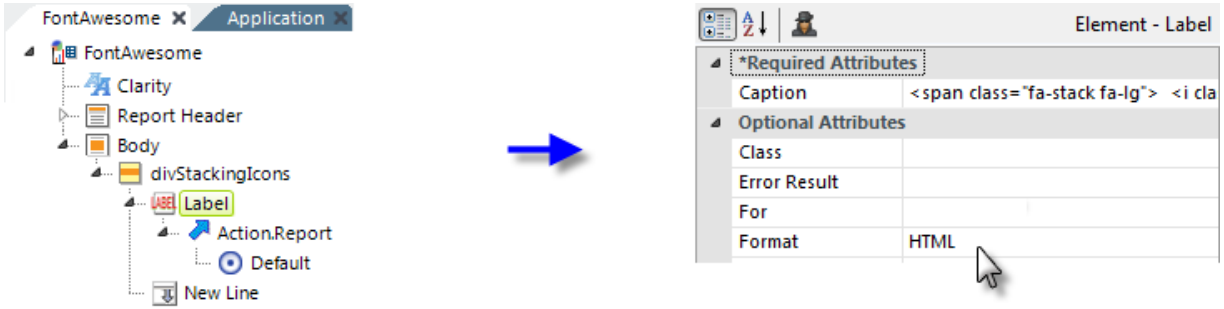


And the results are shown above.

The height of the tabs and the size of the text is being controlled by the *height* of the icon + the padding.

Stacking Icons

Font Awesome has included CSS to stack icons, allowing you to combine them. In order to do this, instead of inserting the Font Awesome CSS into a **Label** element's **Class** attribute, write the full HTML code into its **Caption** attribute and set its **Format** attribute to *HTML*.



An example is shown above. We've used a Label element and set its Caption attribute value to:

```
<span class="fa-stack fa-lg">
 <i class="fa fa-square-o fa-stack-2x"></i>
 <i class="fa fa-bars fa-stack-1x"></i>
</span>
```

The *fa-stack* class is an FA class that stacks icons. In this example, we've stacked a "bars" icon on top of a "square" icon. Note that we used `<i></i>` tags, without any actual text, for convenience only; you can also use other tags.

And the resulting icon is shown above.

Customizing Font Awesome

You can download and modify the un-minified version of `font-awesome.css` if you want to make your own customized version of it.

Themes

Our stock **themes** are designed to help developers instantly apply a consistent "look" to their applications.

The following topics discuss themes, how to apply them, and how to create them:

- Selecting a Theme in Studio's Wizard
- Applying Themes Manually
- Themes and Style Sheets
- Creating Your Own Themes
- Theme Release Notes

About Themes

Several **standard themes** are included with Logi Info for your use. These themes include a collection of graphic images, a style sheet, and a template modifier file, which impart a specific appearance to a Logi application.

You can view the currently available themes in action in the Sample Themes Library. Additional Sample Theme Applications can be found on our Sample page.

Themes do the work for you, setting appearance attributes for charts and Data Tables, and for complex super-elements such as the Analysis Grid, generally making it easy to produce great looking reports without an in-depth knowledge of Cascading Style Sheets. You can easily switch between themes in order to experiment with them.

As of this writing the standard themes include:

Theme	Description
Arizona	This theme, introduced as an independent download in May 2017, has earthy tones in the grand tradition of the American Southwest. See the Sample Themes library on DevNet to download it.
Black Pearl	A black background and dramatic colors make this an unusual theme.
Clarity	Offers a clean look that's "easy on the eyes".
Professional Blue	Gradients and a polished look, with colors grouped in the blue spectrum. *This theme is no longer included, starting with v12.1.*
Professional Green	Gradients and a polished look, with colors grouped in the greenspectrum. *This theme is no longer included, starting with v12.1.*
Signal	A modern, professional, and attractive look that works well with both desktop and mobile displays.
Silver	A simple, understated look.
Simple Blue	A stronger appearance with flat blue color scheme. *This theme is no longer included, starting with v12.1.*
Transit	A bold theme designed to look good on mobile devices.

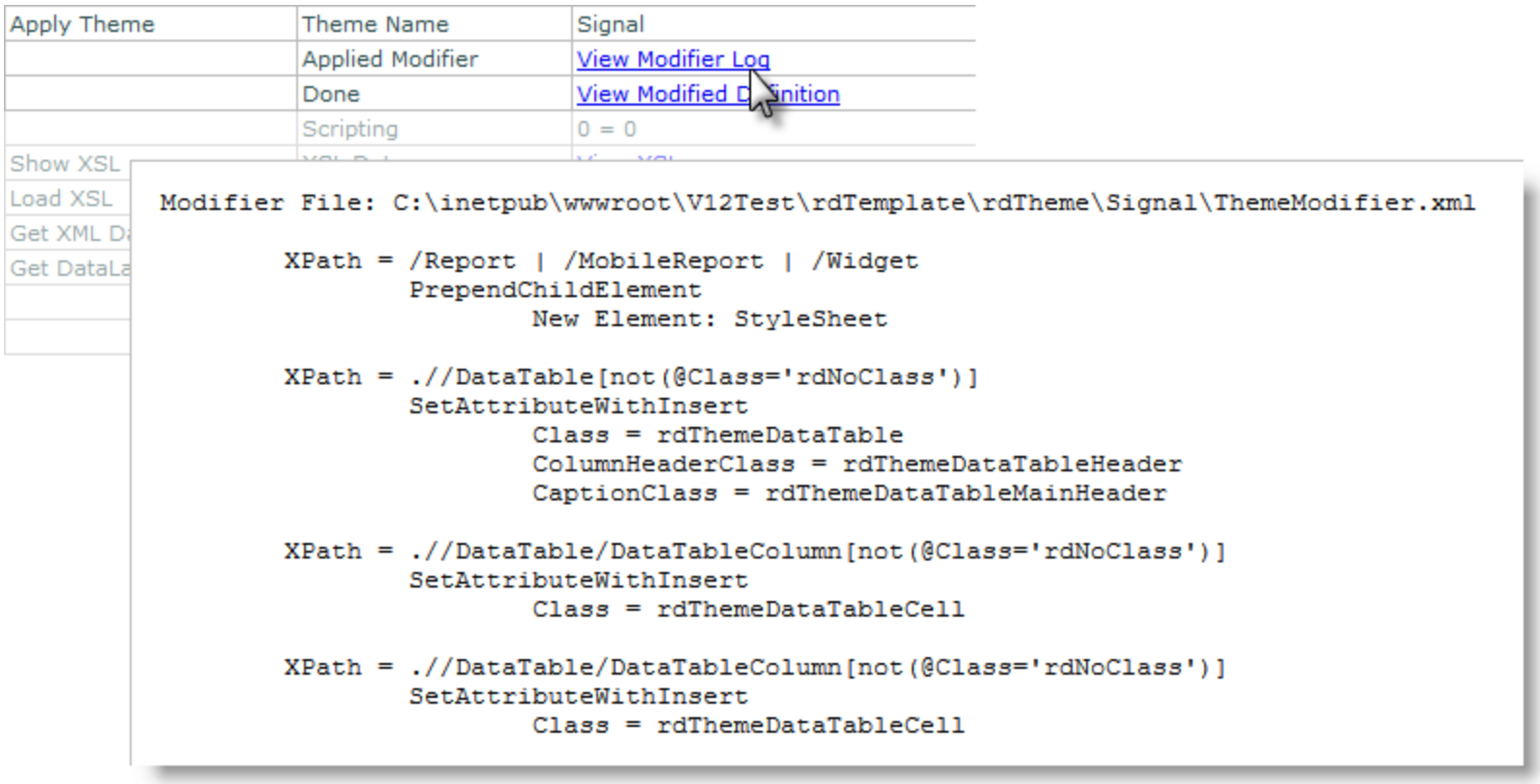
Standard themes *may be updated* when a new Logi Info version is released and this may produce changes in the appearance of applications that are upgraded to the new version. See "Theme Release Notes" on page 249 to determine when the standard themes have been updated.

Uses Other Than Appearance

Themes, however, are not "just a pretty face". Because themes are capable of altering your report definitions, they can be used for purposes unrelated to application appearance. For example, they can be used to set default attribute values. This can be very useful if you consistently use the same set of attribute values for, say, Data Tables, and you want to apply them as defaults every time you start a new definition or application.

Themes in the Debugger Trace Page

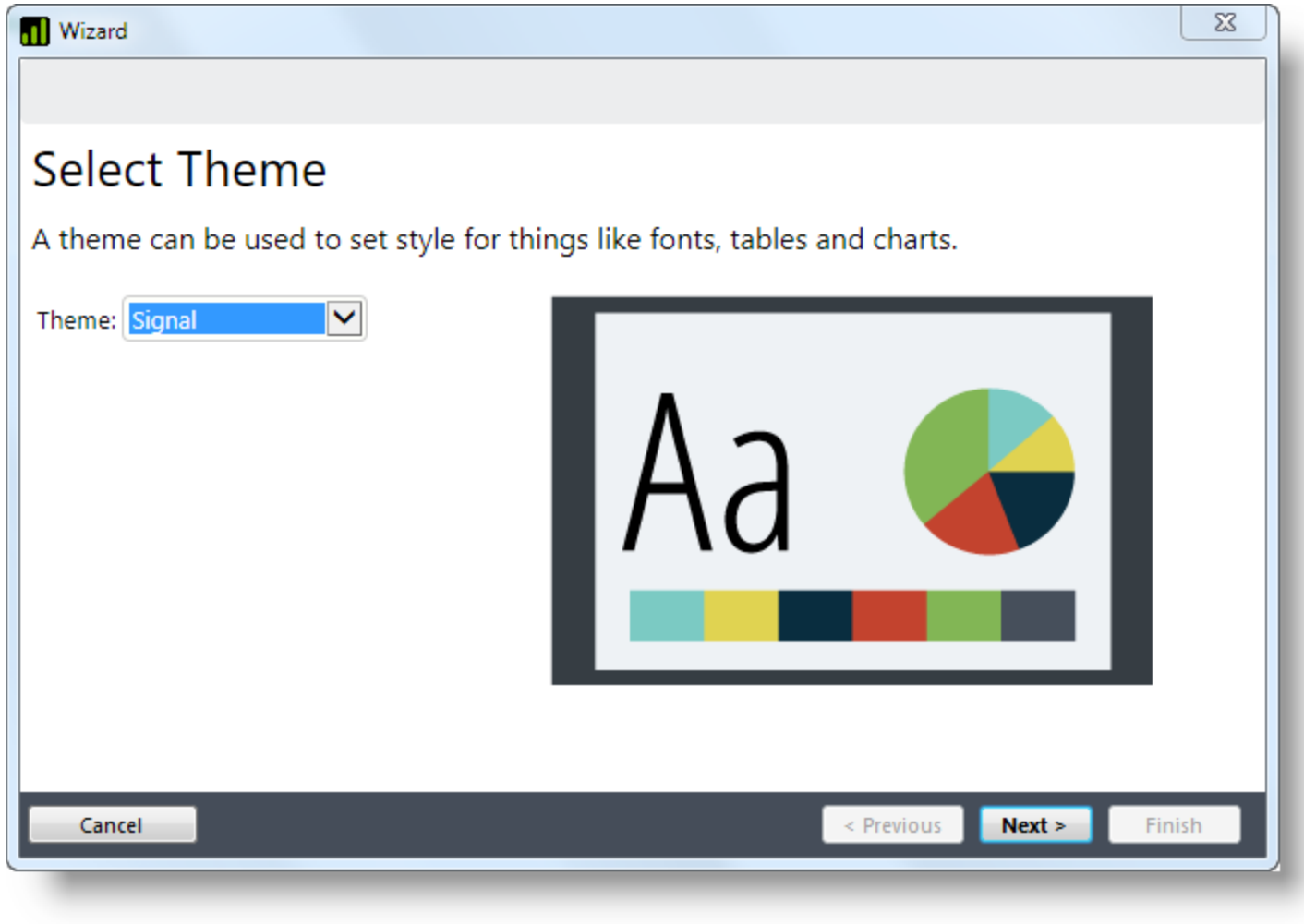
Themes use one or more Definition Modifier Files (DMFs) to alter the definition and, when debugging links are turned on, special entries will appear in the Debugger Trace report (see *Debug Reports*):



Click the entries shown above to view the DMF log and the source code of the modified definition. This information can help you understand how a theme works and may be helpful if you want to create your own custom theme.

Selecting a Theme in Studio's Wizard

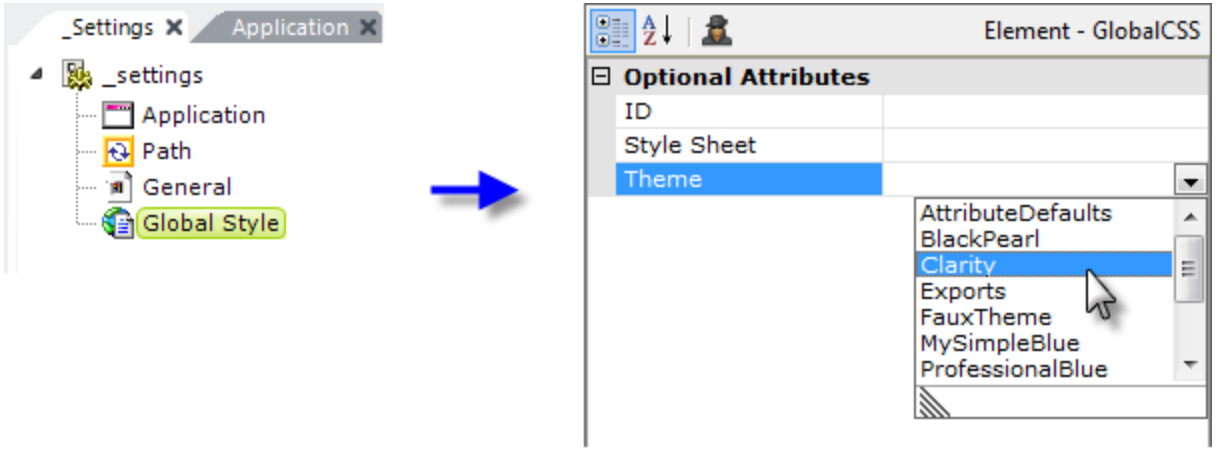
When you use the **New Application** and **Select Theme** wizards in Studio, one of the steps is the selection of a theme:



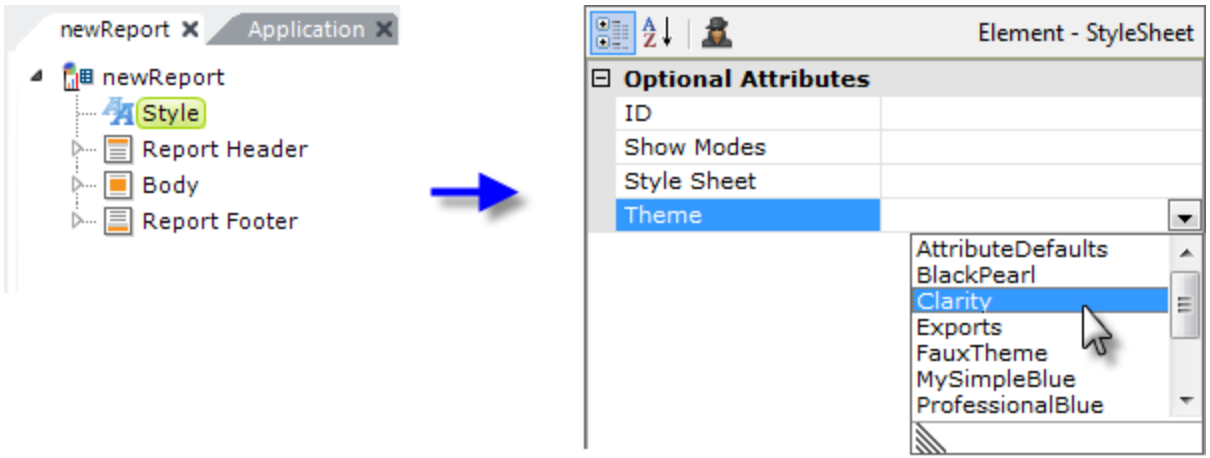
Just select a theme from the list and it will be applied automatically to your new application.

Applying Themes Manually

You can also apply a theme manually. A theme can be applied at the application level (which is what the wizard does):



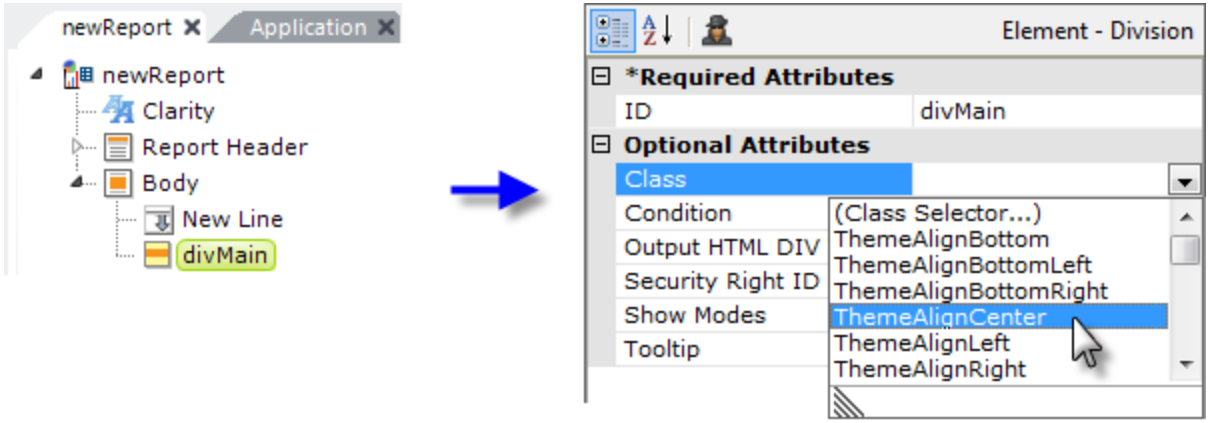
As shown above, to apply a theme to the application as a whole, add a **Global Style** element to the _Settings definition. Then set its **Theme** attribute to the name of one of the available themes. All reports in the application will have the theme applied to them. To apply a theme to an individual report definition:



As shown above, add a **Style** element to the report definition. Then set its **Theme** attribute to the name of one of the available themes. The report will have the theme applied to it.

Themes and Style Sheets

When a theme is selected for a definition or application, it automatically applies a specific appearance to super-elements, like the Analysis Grid. In addition, the theme makes available a standard set of **styleclasses** that developers can apply to regular elements. This can save a lot of time and helps to create a consistent appearance.



In the example shown above, we see a definition that has had a theme applied to it. A **Division** element has been selected and one of the theme-supplied style classes is being assigned to it. As you can see, all of the theme-supplied classes begin with the letters "Theme" and they address alignment, font sizes, borders, and other useful styles.

Themes do not set the style class for *every* element; for example, Label and Rows, Row, and Column Cells (HTML tables) are not affected. You can set the **Class** attribute for these elements independently of the theme.

You can specify both a style sheet *and* a theme in a Global Style or Style element. If you have a style sheet and a theme assigned, the available classes from both will appear in the drop-down class list.

Interaction with Other Style Sheets

You may already have a style sheet that you like to work with when developing reports. How will that **interact** with a theme when it's applied to your report?

If you have assigned a style class to an element and the theme also applies a style class to that same element, the theme's style will be applied first, then yours. This means both styles will be applied but your style will "win" when there's a direct conflict. When there's no conflict, the effect will be *additive*.

Themes and Exports

Theme effects will also be applied to exports, which may or may not be desirable, and there is no way to block or undo the effects of a theme that has been assigned using the Global Style element. So if you plan to export reports to a format where appearance is important, such as PDF, you may want to assign themes using individual Style elements in each report, instead of the Global Style element.

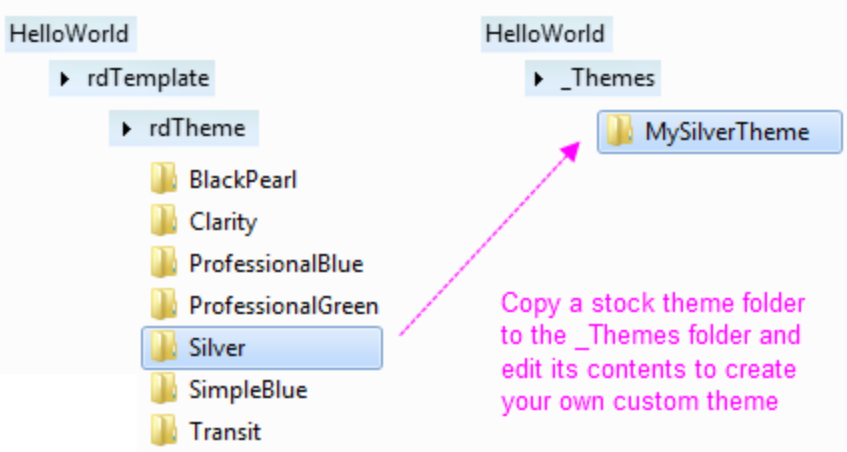
Creating Your Own Themes

Beginning with v12.1, Logi Studio includes the **Theme Editor** wizard, a tool that allows you to quickly and easily create your own custom themes. This tool is also available as an *element*, if you'd like to build an application that lets administrative users modify the application's appearance at runtime. For more information, see "Use the Theme Editor" on page 253. Developers working with earlier Logi Info versions can create custom themes manually. You're encouraged to examine the standard themes and use them as the basis for creating your own themes. The remainder of this topic discusses this process.

Creating a Custom Theme Manually

At a minimum, themes consist of a Theme Modifier file, which does the work. Themes that impart an appearance generally also include a collection of images and a theme style sheet. Each theme is contained in its own folder, which bears the name of the theme. You can view the currently available themes in action in the Sample Themes Library, as well as the Sample Themes Application.

When you create a Logi application, the standard themes distributed with Logi products are stored in separate folders in the `<yourApp>\rdTemplate\rdTheme` folder; these should *never* be edited.



However, you can copy standard theme folders and then customize the files within them to create your own custom theme. Copy the standard theme *folder* to the _Themes folder beneath your application folder, i.e. `<yourApp>\_Themes`, (which you may have to create) and rename it to the name of your new custom theme, as shown above.

Apply your customizations to the files in this folder. Your custom theme will then be available for your application, by name, along with the standard themes, in the Style and Global Style elements' **Theme** attribute selection list, and will also be available for deployment with Studio's Deployment Tool.

One of the key mechanisms in a theme is the "template modifier file" or TMF. In your copied Theme folder, you'll find such a file, named `ThemeModifier.xml`. A TMF is an XML file that includes instructions for modifying the elements and attributes in a report definition. The TMFs in the standard themes are quite complex, as they attempt to cover *all* of the elements available for use in a report. Your custom TMF doesn't necessarily have to be so all-encompassing; it could, for example, only address the charts or other elements that are specifically used in your reports.

If you examine the TMF and style sheet in one of the standard themes, you'll see that the class and ID names are all carefully constructed to be unique, which is very important and which you are also encouraged to do.

 The standard `ThemeModifier.xml` file may contain references to the standard style sheet and standard images in the `rdTemplate/rdTheme/<Theme>` folder. In your custom TMF, you may need to change these to reference the style sheet and images in your custom Theme folder.

Template Modifier Files provides detailed information about creating and using TMFs.

DevNet's Element Gallery page has information about all of the elements and their attributes and may be useful when creating your own themes.

Your Themes in the New Application Wizard

If you want your custom theme to be available in Studio's **New Applicationwizard**, copy its folder to (you may have to create the _Themes folder):

```
(.NET) C:\Program Files\LogiXML IES Dev\LogiStudio\BaseApp\<version>\_Themes
(Java) C:\Program Files\LogiXML IES Dev\LogiStudio\BaseApp\<version>\_Themes
```

The New Application wizard displays a thumbnail image of each theme and, if you want to include images for your custom theme, place it, as a 225x160px .JPG image with the same name as your theme, in this folder:

```
C:\Program Files\LogiXML IES Dev\LogiStudio\bin
```

Finally, If you create a custom appearance-specific theme that you think looks cool, it would be great if you **shared** it here on DevNet.

Element-Specific Theme Classes

Theme definitions also include an element-specific **class filtering** mechanism. This mechanism ensures that theme-related style classes meant for a specific element will appear in the Class attribute drop-down list for that element and *only* for that element. If you're creating your own theme, use the following text in its Theme.css file to indicate the start of those class definitions:

```
/*User classes*/ /*rdElement: elementName */
```

where *elementName* is the name of the element (the XML source name, which appears at the top of the Attributes panel, not the name that appears in the Element Tree, which may have embedded spaces)

Classes that apply to **multipleelements** can be specified by using the | character to combine the element names, like this:

```
/*rdElement: element1Name | rdElement: element2Name */
```

Classes defined like this will appear in the class list for the named elements, along with all the other appropriate theme classes and style sheet classes.

Here's a more complete example (don't enter the directions, within square brackets):

```
[ add after all other standard classes, just before end of file ]

/*User classes*/
/*rdElement: Label */
.font12pt {
 font-size: 12pt;
}
/*rdElement: Division */
.font16pt {
 font-size: 16pt;
}

/*rdElement: InputText | rdElement: InputPassword*/
.alignMiddle {
 vertical-align: middle;
}
/*End Element*/ [ this is already in the file ]

Once the custom theme is applied, the class "font12pt" will appear in the list of Class attribute
```

choices for all Label elements, and "font16pt" will appear in the list of class choices for all Division elements. "alignMiddle" will appear in the list of choices for both the Input Text and Input Password elements.

Theme Release Notes

This table shows the history of changes to the Logi Infothemes:

Version	Date	Changes
12.2 SP 6	May 2017	New **Arizona** theme introduced.
12.1.188	Feb 2016	**Pro Blue**, **Pro Green**, and **Simple Blue** themes are no longer distributed. They're still supported in an application but cannot be edited with the Theme Editor. The *styling* of the other standard themes has been updated, changing their appearance.
11.3.049	30 Jun 2014	New **Signal** theme introduced.
10.2.424	7 Sep 2012	Original set of standard themes updated; **Clarity** and **Transit** themes introduced.
10.0.31	4 Feb 2010	Original set of standard themes introduced.

Theme Editor

The **Theme Editor** allows you to easily create custom themes based on the standard themes distributed with Logi Info.

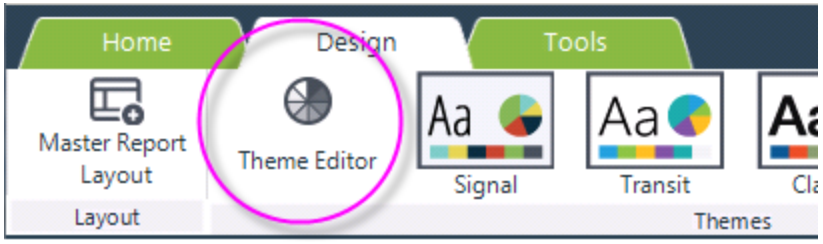
The following topics discuss the Theme Editor:

About the Theme Editor

Themes allow you to apply a consistent appearance to your Logi applications. Several standard themes are included with Logi Info for your use. Themes do the work for you, setting appearance attributes for charts and Data Tables, and for complex super-elements such as the Analysis Grid, generally making it easy to produce great looking reports without an in-depth knowledge of Cascading Style Sheets. You can easily switch between themes in order to experiment with them. For more information, see "Themes" on page 235. The **Theme Editor**, available in Log Info v12.1+, allows you to create your own *custom* themes by modifying and saving a standard theme. Colors, typography, and other theme styling can be configured to achieve the special "look" you desire. The Theme Editor is available for use in two ways:

As a Logi Studio Wizard

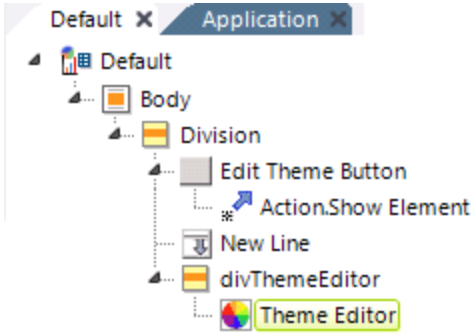
Logi Studio offers the Theme Editor as a *wizard* for developer use. Once you open an application in Logi Studio, the main menu's Design tab includes a Theme Editor icon:



Selecting this icon launches the wizard and prompts you to select one of the standard themes, or one of your existing custom themes, if any, for editing. You're not allowed to directly edit the standard themes so, if you select one of them, the wizard will make a copy of it for you to edit and prompt you for a name for your new custom theme. Your custom theme will be saved in your application's _Themes folder and you'll be prompted to apply this theme, in the form of a**Global Style** element in the application's _Settings definition.

As a Logi Info Element

Logi Info also makes the **Theme Editor** available as an *element* that can be included in your Logi application. This allows you to create an application that allows administrative users to modify its appearance at runtime, by editing a custom theme you've created. However, users cannot use this element to *create* custom themes.



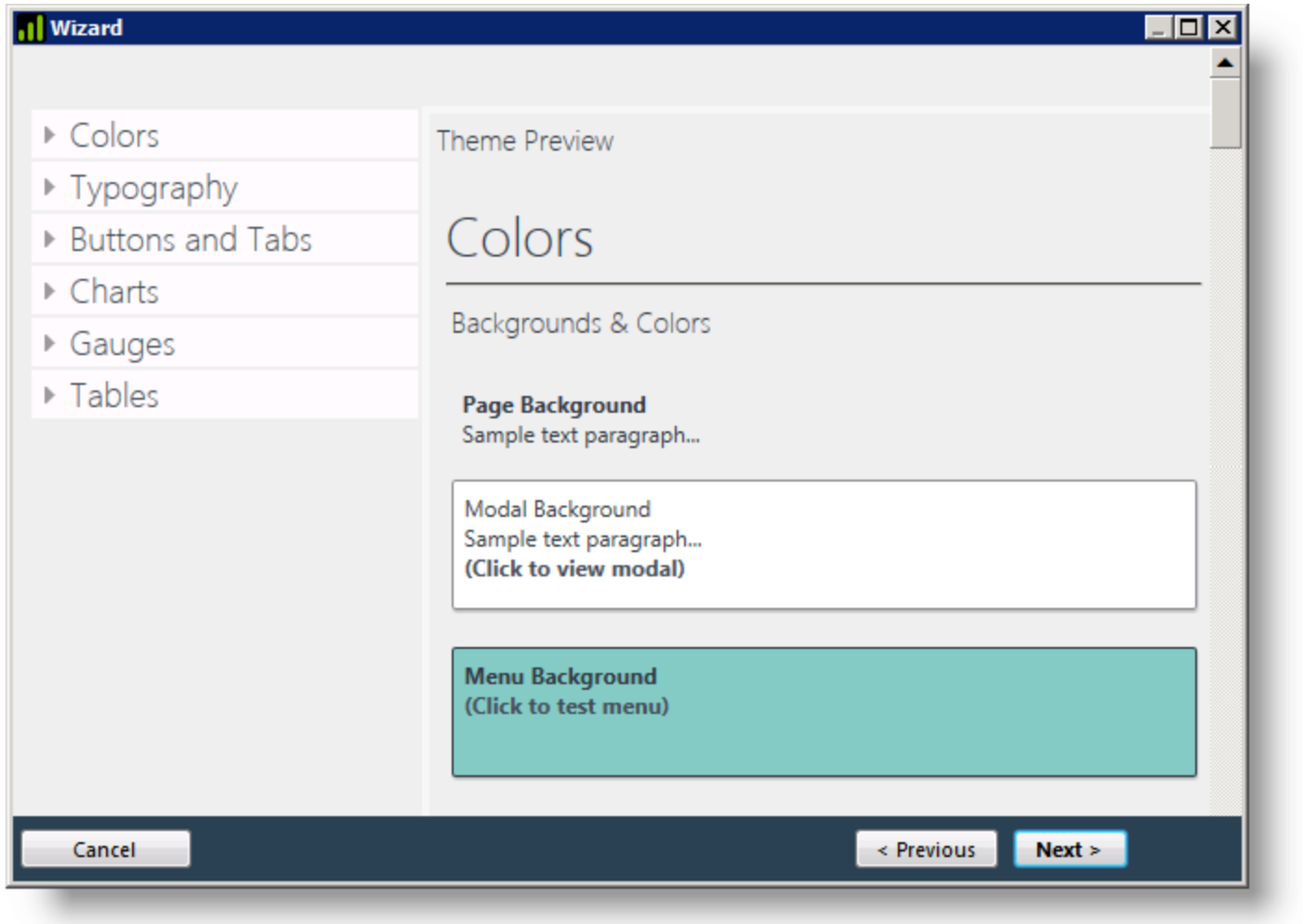
The Theme Editor element's runtime availability can be controlled using its **Security Right ID** attribute (when using Logi Security) or by placing it beneath a container element, such as a Division, that has a Condition or Show Modes attribute. In the example shown above, a Button and Action.Show Element are used to show/hide a Division containing the Theme Editor. The element's only required attribute, **Theme**, identifies the name of the custom theme to be edited. An optional attribute, **Autosave Theme**, controls whether a Save button is displayed (the default) or whether each configuration change is automatically saved as it's made (the Save button is not displayed).

Legacy Custom Theme Considerations

Custom themes created from standard themes distributed with versions of Logi Info prior to v12.1 *will* still work correctly with your Logi v12.1+ applications. However, v12.1+ standard themes use a different internal structure, which is the one recognized by the Theme Editor. Therefore, custom themes based on standard themes distributed with earlier Info versions cannot be edited with the Theme Editor. You will have to recreate them using v12.1+ standard themes in order to be able to edit them with the Theme Editor. The Theme Editor does not affect any custom code entered into a custom theme's Template Modifier File. If you choose to recreate a custom theme built on a pre-v12.1 standard theme, you will need to manually copy your TMF code into the new custom theme's TMF. This also applies to any new, non-standard style classes you may have added to the custom theme's CSS file.

Use the Theme Editor

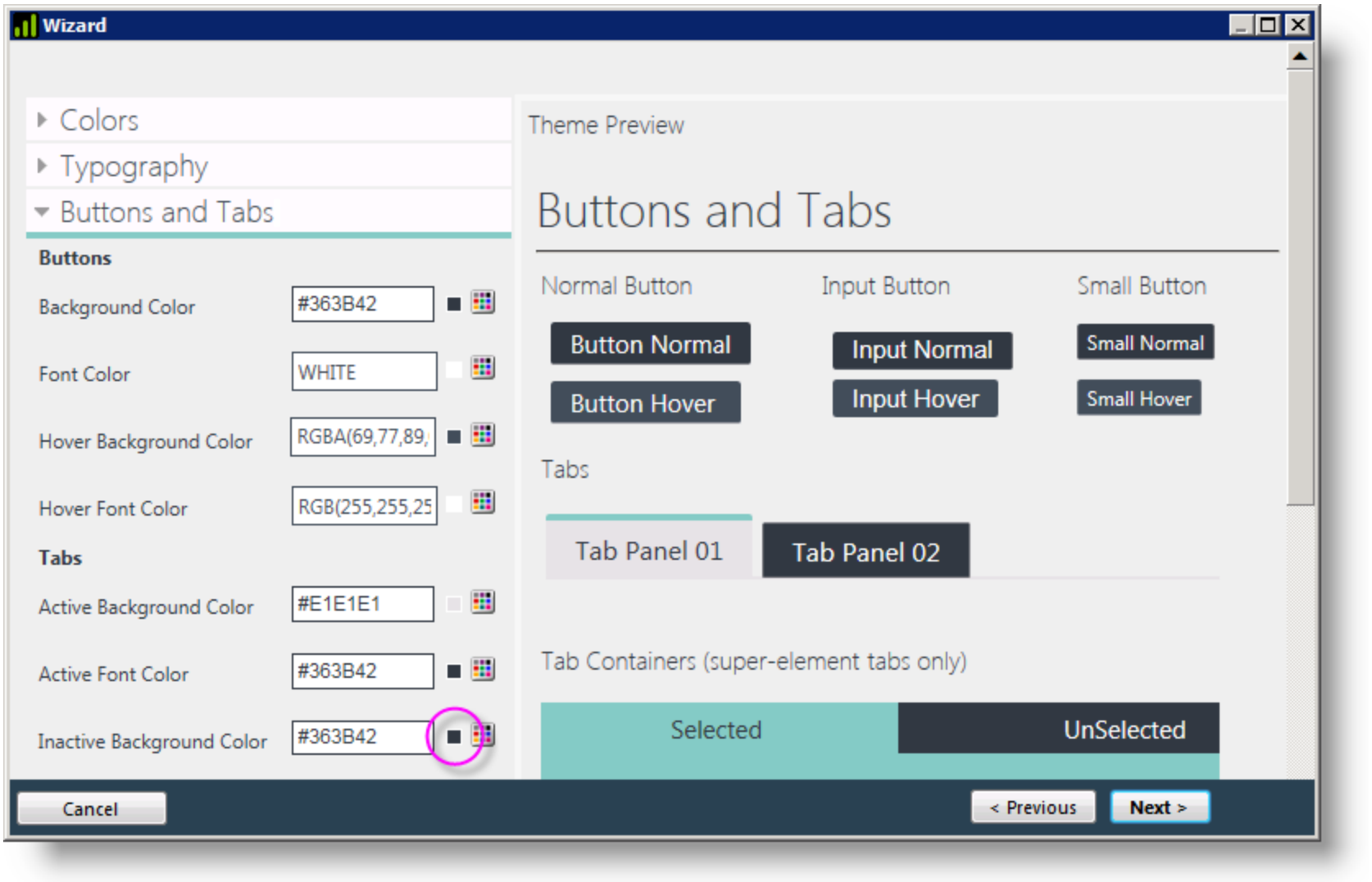
The Theme Editor provides a series of controls that let you configure your custom theme:



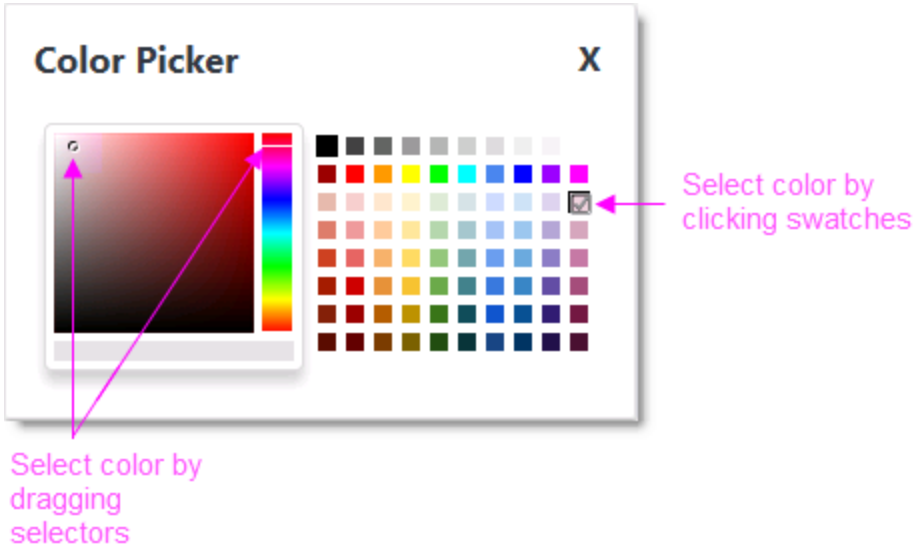
The Theme Editor *wizard* interface in Logi Studio is shown above and it includes:

- **Styling Categories** presented as expandable sections, on the left.
- A **Preview** panel that immediately shows the effects of any changes, on the right.

When the Theme Editor *element* is used, the categories and previewappear on the report page rather than in a dialog box and there are no Cancel, Previous, orNext buttons. The element can be configured so that a **Save** button is displayed,or it can be configured to automatically save each change as it's made.


Wizard
Colors
Typography
Buttons and Tabs
Buttons
Background Color
#363B42
Font Color
WHITE
Hover Background Color
RGBA(69,77,89,
Hover Font Color
RGB(255,255,25
Tabs
Active Background Color
#E1E1E1
Active Font Color
#363B42
Inactive Background Color
#363B42
Theme Preview
Buttons and Tabs
Normal Button
Input Button
Small Button
Button Normal
Button Hover
Input Normal
Input Hover
Small Normal
Small Hover
Tabs
Tab Panel 01
Tab Panel 02
Tab Containers (super-element tabs only)
Selected
UnSelected
Cancel
< Previous
Next >

In the image above, the **Buttons and Tabs** category has been expanded and the controls are visible. Color controls allow you to enter a color by name, as a decimal RGB value, or as a hex RGB value preceded by a pound sign, e.g. #112233. The currently-selected color is shown by the small square (circled above) next to the Color Picker icon.



You can also click the **Color Picker** icon to open a pop-up panel with a palette of color selections, shown above. Once you've made alldesired changes, click the **Next** button (or, when using the Theme Editor element, click **Save** - unless auto-save mode has been enabled, as mentioned earlier). You can click the **Cancel** button to abandon any unsaved changes.

Under the editor's *Buttons and Tabs* category, in the Tabs section, the Container Font Color and Container Background Color items refer to the tabs that can be seen in super-elements, such as the Analysis Grid and the Dashboard. They do not refer to the stand-alone Tabs and Tab Panel elements.

Editor Items vs. Theme Classes

Most of the items in the Theme Editor are easily correlated to the elements and classes in your report definition. The following table helps you correlate other items you see in the editor to the actual Theme class names you'll see in Logi Studio when selecting values for an element's **Class** attribute.

Theme Editor Item	Element "Class" Attribute Option
CB Border Color	ThemeContainerBordered
CS Background Color + CS Font Color	ThemeContainerShaded
CSB Background Color + CSB Font Color	ThemeContainerShadedAndBordered
Error Text Font Color + Error Text Background Color + Error Text Border Color	ThemeErrorText
Heading 1	ThemeHeaderLargest
Heading 2	ThemeHeaderLarger
Heading 3	ThemeHeaderLarge
Heading 4	ThemeHeader
Heading 5	ThemeHeaderSmall
Heading 6	ThemeHeaderSmaller

Theme Editor Item	Element "Class" Attribute Option
Heading 7	ThemeHeaderSmallest
Text Negative Font Color	ThemeTextNegative
Text PositiveFont Color	ThemeTextPositive

Share Your Custom Themes

Have you created a cool custom theme? Would you like to share it with other Logi Info developers? Zip your custom theme folder and email it as an attachment to DevNet@LogiAnalytics.com and we'll add it to the Samples section on DevNet and give you proper credit. When you submit a custom theme to DevNet, you agree that it is freely made available to other developers without any restrictions or attribution requirements. Submissions are covered by the DevNet Terms of Use.

Internationalization and Localization

Internationalization and localization are means of adapting Logi applications to the different languages, regional differences, and cultural requirements of a specific user community.

The following topics discuss issues related to these challenges:

About Internationalization and Localization

Internationalization is the process of designing an application so that it can be adapted to various languages and regions without low-level engineering changes. *Localization* is the process of adapting an internationalized application for a specific region or language by adding cultural- or locale-specific customizations and translating text. Together these are often referred to as *globalization*. *Culture* settings, which are generally governed by the client computer operating system and browser, are used to control the visual display of:

- Language, including fonts, font-sizes, symbols, reading direction, etc.
- Number formatting, including thousands-place and decimal separators

- Date & time formatting, including the use of different calendar formats
- Time zone
- Currency formatting, including symbols and positions of currency markers
- Weights and measures
- Paper size

The culture name, used to specify culture settings, is a combination of a two-letter lowercase culture code (associated with a language) and a two-letter uppercase subculture code (associated with a country or region). The format for a culture name is:

```
languageCode-country/regionCode
```

Examples include `ja-JP` for Japanese (Japan) and `en-US` for English (United States). In rare cases where a two-letter language code is not available, a three-letter code is used. Not all culture names are available on all operating systems; for example, `en-SG` (English-Singapore) is not available on Windows XP. Developers may rely on the fact that Logi products use technology based on internationally-accepted standards, such as XHTML, XML, and CSS. There are no language- or culture-specific versions of Logi products and *internally* they can be expected to operate consistently without regard to language or culture. For example, there is no special set of elements for one language and a different set for another language. It is the *output* generated by Logi products, reports and web pages, that can be customized by globalization. Logi Analytics products include features designed to make globalization possible and to support a variety of cultures. Developers have successfully configured their Logi applications for use in numerous countries, languages, and cultures. This useful presentation, *Designing for International Users*, from the W3C, discusses many of the generic issues associated with developing globalized applications. The following sections address specific areas of interest for developers who want to globalize Logi applications.

Datasource Language

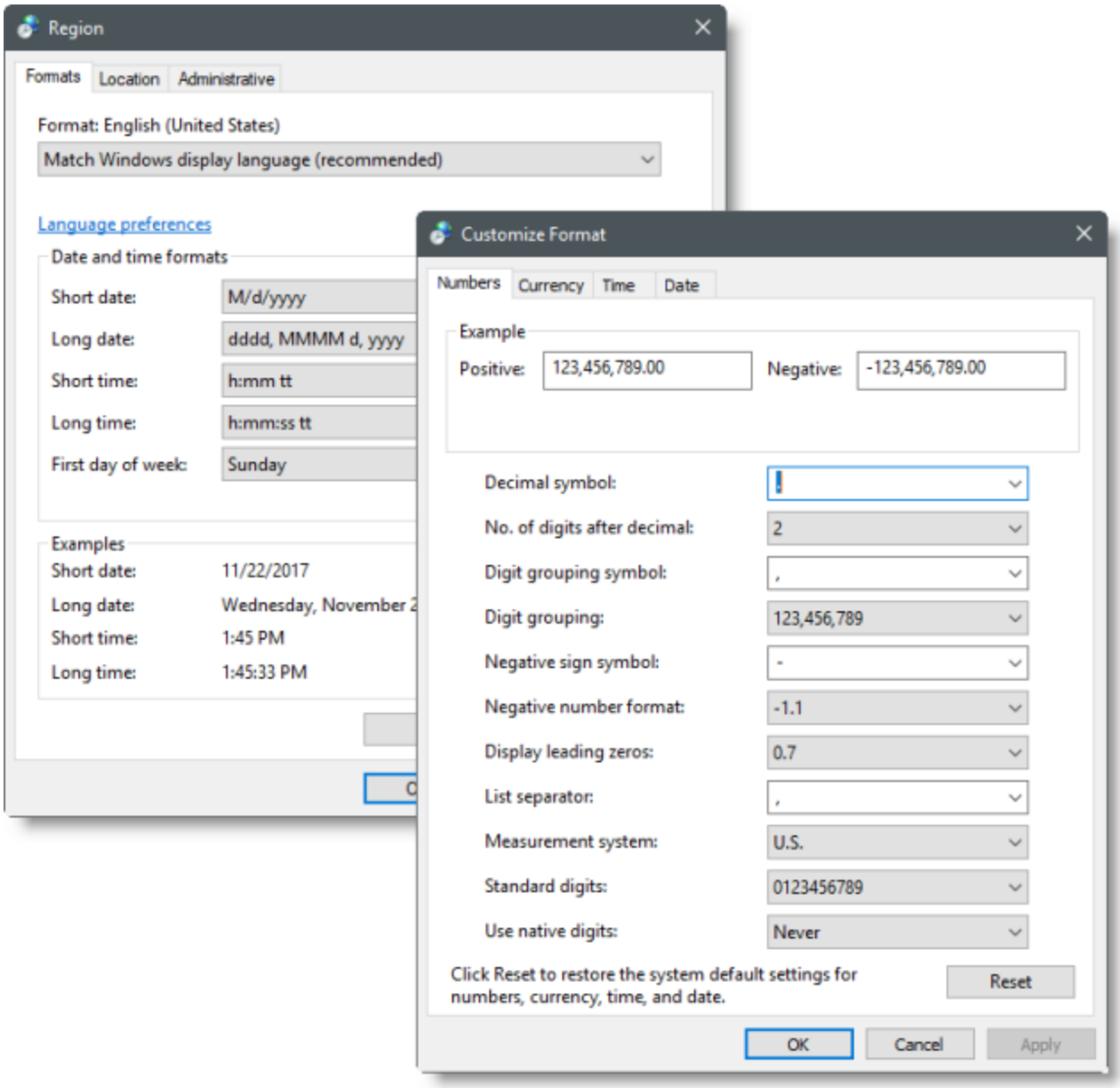
Data displayed in Logi reports is typically stored in datasources external to the Logi application and these datasources may have their own language- or culture-specific configuration possibilities. This is significant for the developer and may require coordination with IT system administrators to ensure proper configuration. In particular, database servers often have language configuration options or are available in language-specific product versions. The specifics of these are beyond the scope of this topic, but system and database administrators involved with databases providing data for globalized Logi applications need to concern themselves with issues such as:

- Character encoding (Unicode, UTF8, etc.)
- Collation
- Case-sensitivity
- Accent-sensitivity

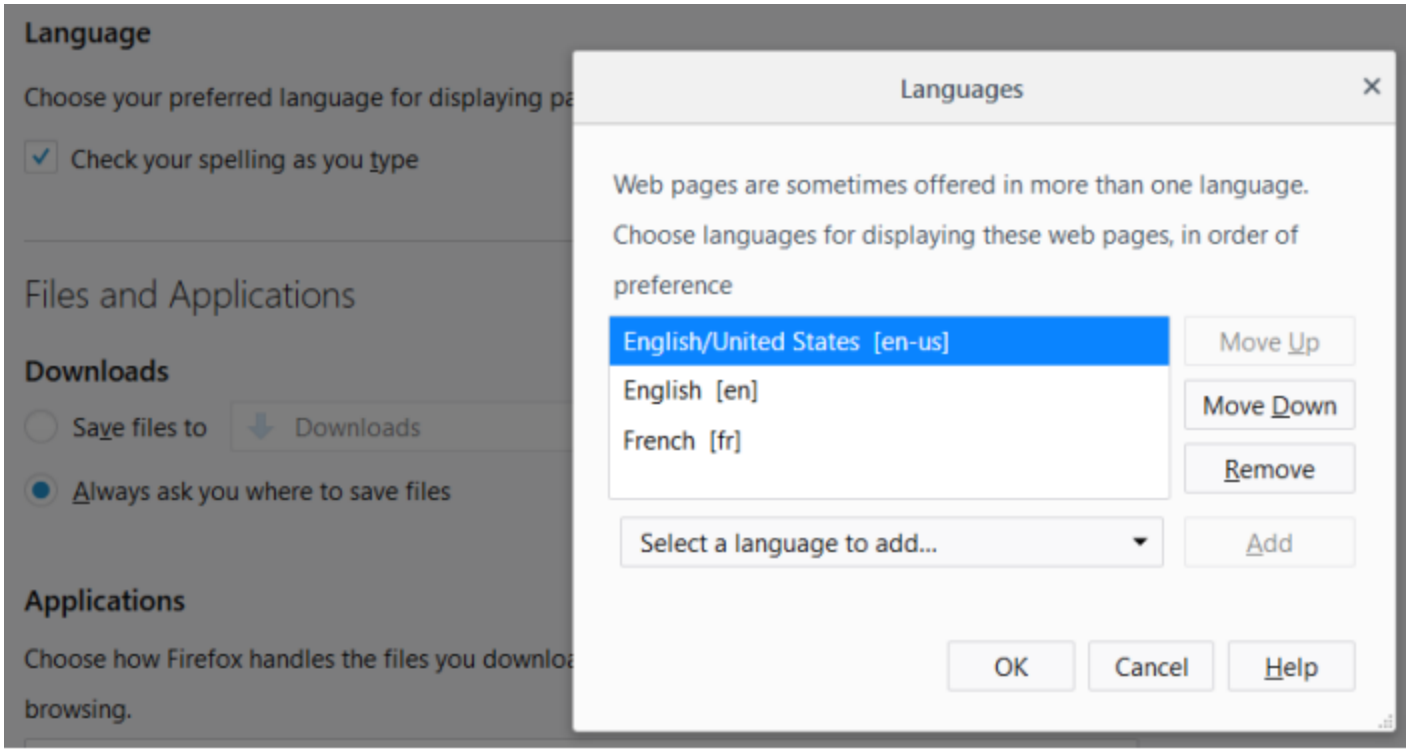
Database server vendors often have extensive information available about their products and globalization. Here are some vendor resources: Microsoft - *International Considerations for SQL Server 2008 R2*
Oracle - *Database Globalization Support Guide for 11g*

OS and Browser Culture Configuration

Logi applications are viewed on a client computer or a mobile device, with a browser running on top of an operating system (OS). Both the OS and browser have culture-related settings that may need special configuration for use with globalized applications.


Region
Formats
Location
Administrative
Format: English (United States)
Match Windows display language (recommended)
Language preferences
Date and time formats
Short date:
M/d/yyyy
Long date:
dddd, MMMM d, yyyy
Short time:
h:mm tt
Long time:
h:mm:ss tt
First day of week:
Sunday
Examples
Short date:
11/22/2017
Long date:
Wednesday, November
Short time:
1:45 PM
Long time:
1:45:33 PM
Customize Format
Numbers
Currency
Time
Date
Example
Positive:
123,456,789.00
Negative:
-123,456,789.00
Decimal symbol:
No. of digits after decimal:
2
Digit grouping symbol:
Digit grouping:
123,456,789
Negative sign symbol:
Negative number format:
-1.1
Display leading zeros:
0.7
List separator:
Measurement system:
U.S.
Standard digits:
0123456789
Use native digits:
Never
Click Reset to restore the system default settings for numbers, currency, time, and date.
Reset
OK
Cancel
Apply

In the Windows 10 operating system, for example, culture settings are referred to as **RegionalSettings** and can be set, as shown above, via the Settings applet. Other operating systems have similar settings. The system-wide formatting configured here is used by the **Format** attribute of various Logi elements to format items in reports (discussed in more detail below). Your OS will also have setting for the default input *language* for the system.



Browsers have language settings, as shown in **Firefox** above, which are generally available under browsers' Tools or Options menu. Most web pages, and Logi reports, contain information that tells the web browser what language and character set to use when displaying the page. This type of information is referred to as *language encoding* and is derived from the OS settings. You

can, however, *override* the OS settings and force the browser to use a different language. When you change the browser's language here, a number of formatting changes will happen *automatically*. For example, selecting `French [fr]` and making it the first language of preference in the example above, will cause

`$1,234.00` to appear as `€1.234,00`

The currency format is changed so that the Euro symbol replaces the Dollar Sign, and the thousands and decimal separators are changed to those used in France. In this way, reports hosted on servers in one country can be automatically reformatted for viewing in another country. You may encounter references to an "invariant culture" - this is a term Microsoft uses to identify use of the English language without a specific geographic association. For Logi applications, this refers to English and the United States or `en-US`. The culture setting configured for a browser is available in a Logi application using the @Function.UserCulture~ token. This token can be used in several elements to modify their behavior.

Globalization and CSS

In Logi reports, stylesheets and their classes can be used to control changes such as fonts, font sizes, line heights, and reading direction when language changes are necessary. This can be particularly useful when dealing with languages like Chinese, where users tend to prefer different fonts, and can also be useful to better harmonize the look of mixed, script-specific fonts, such as when mixing Arabic and Latin fonts. In general, the use of CSS selectors such as `text-align`, `position`, `background-position`, `float`, and `clear`, which often use absolute positioning, need to be reviewed when globalizing an application. CSS also offers the `lang` selector, which can be used in a variety of ways to affect the display of text in different languages. In addition, CSS file encoding can be set by use of the @charset command at the top of the file, for example:

```
@charset "iso-8859-1"
```

The standard themes provided with Logi reporting tools include stylesheets and developers may need to create their own custom globalized themes, duplicating all but the stylesheet files, if themes are being used. The stylesheet files would then be customized for globalization. For more information about creating custom themes, see "Themes" on page 235.

Report Page Layout Considerations

Special care must be taken when designing reports that will be globalized, especially if they include user input elements. For example:

- **Avoid using fixed widths and heights**. When elements include sizing attributes, use a percentage, rather than a fixed number of pixels, whenever possible.
- **Plan for the possibility of text-wrapping**. Labels, captions, and other text may become longer when displayed in some languages, overrunning or wrapping within the space provided for them.
- **Separate captions from input elements**. Use Row, Rows, and Column elements to create your own HTML table to contain input elements, rather than an Input Grid. Use separate Label elements for input element captions rather than using their Caption attributes.

In general, any physical boundaries on the report page that may be affected by the size and length of text or data, which may change with different cultures,need to be given special consideration.

Using the Globalization Element

The _Settings definition in a Logi application provides application-wide configuration of various settings, such as connections, team development, etc. The **Globalization** element, which provides valuable assistance when developing a multi-cultural application, is also available for use the _Settings definition. The following table describes its attributes:

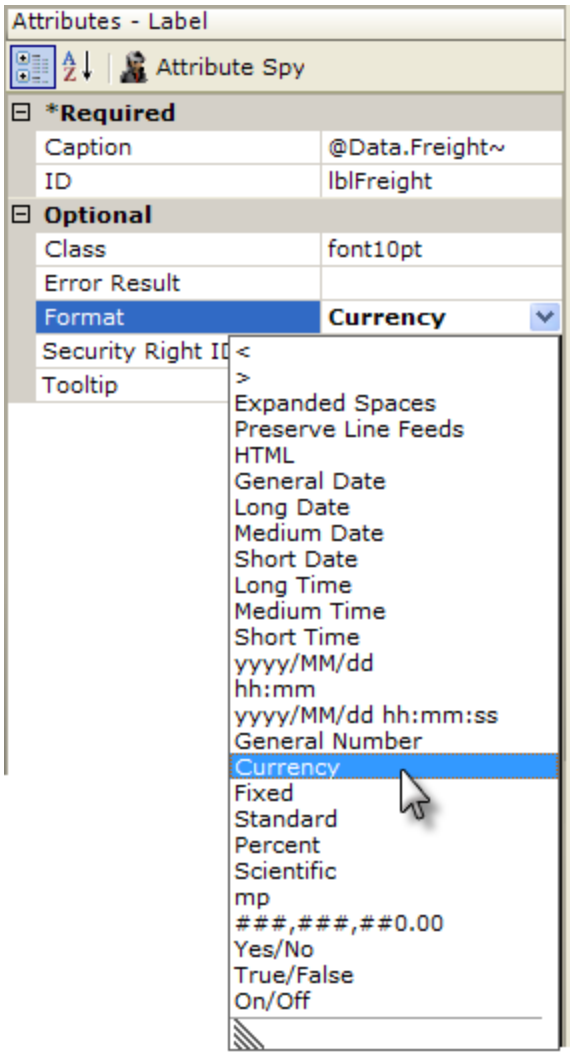
Attribute	Description
Default Input Date Format	Specifies the default input date format used when an Input Date element does not have its own Format attribute set.
Default Input Date Reformat	Sets a default value for all Input Date Reformat attributes. These are available with Input Date elements, which allow the user to enter dates.
Default Input Time Format	The default input time format used when an Input Time element does not have its own Format attribute set.
Default Input Time Reformat	Sets a default value for all Input Time Reformat attributes. These are available with Input Time elements, which allow the user to enter times.
First Day of Fiscal Year	Specifies the first day of the Fiscal Year and is used when calculating dates and quarters for fiscal calendars. The value must be in the format of mm/dd. For example if your fiscal year begins on March 5, enter 03/05. The default value is *01/01*. This value affects the Fiscal Year and Fiscal Quarter formats. See *Format Data*.

Attribute	Description
First Day of Week	Changes the order of days in the week in popup calendars used with date inputs and data calendars. Also changes how the first day of the week is determined for the @Date tokens that return first and last days of the week, such as @Date.ThisWeekStart~ and @Date.ThisWeekEnd~. Valid values are: 0 = Sunday (default), 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday This value affects the Fiscal Week format. See *Format Data*.
Input Reformat Culture	*This attribute is deprecated - do not use it.* A culture value that can be used to reformat input dates and values. You can set this to a different culture so that values from Input elements are returned in that culture. This can be useful when the database expects date and numeric values that are not in the invariant format. In most cases it is best to set up the database or database client account to use an "English" culture. This attribute is not used for dates when Input Date Reformat is used. The culture is specified as a string, such as`en-us`. To see a list of cultures, open IE, select Tools, Internet Options, Languages, then Add. The default culture is the "invariant" culture, which is basically US-English.
Java Font Folder	*Active only for Java applications*. Identifies the folder where TrueType fonts are located for PDF exports. The folder is required for PDF exports with fonts that contain characters that are not in the ISO-8859_1 character set. In Windows this is usually something like `C:\Windows\Fonts`. In Linux, this would be something like`/usr/local/share/fonts/ttfonts`.
Metric Prefix String	Specifies user-defined values to be used when the Metric Prefix ("mp") is selected in **Format** attributes. The range of numbers formatted by metric prefixes isfrom 1000^6 to 1000^-6, so one of 12 characters is applied to each power of 10. The default string of characters is: "a,f,p,n,µ,m,k,M,G,T,P,E" representing atto, femto, pico, nano, micro, milli, kilo, mega, giga, tera, peta, and exa. You can replace this

Attribute	Description
	string with a comma-separated string of your own 12 characters. More information about Metric Prefixes is available here.
User Culture	Normally, the user's culture is obtained from the browser's settings. If a value is supplied here, the User Culture attribute overrides the browser's culture setting with another value. User Culture can be set with a token value, such as @Session or @Request. The culture is specified as a string, such as `en-us`. A list of available cultures can be viewed in the browser, where Language is specified.

Configuring Formatting

A number of individual elements include a **Format** attribute, which formats the data being used. A commonly-used example is the **Label** element. Format attribute values can be entered directly or selected from a drop-down list of formats, which include *Long Date*, *Short Date*, *Short Time*, *Currency*, and *Metric Prefix* (mp).

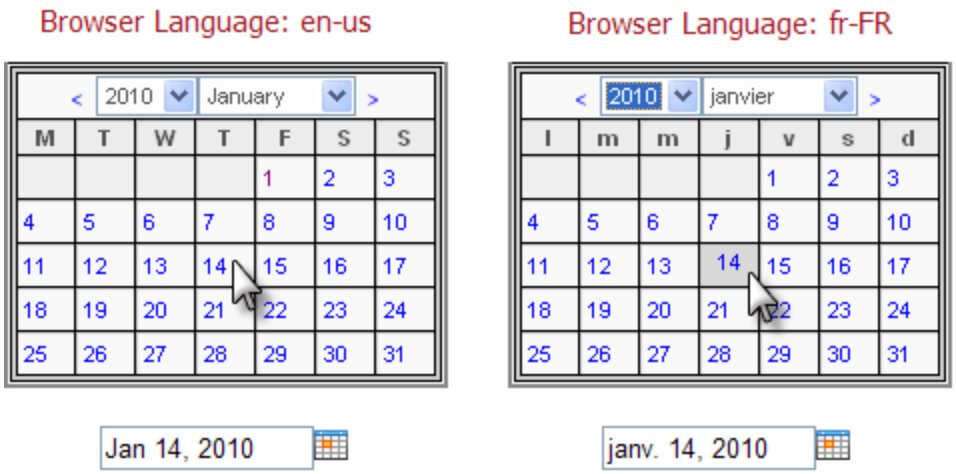


These and other formats are directly affected by the formats set in the Operating System and attributes in the Globalization element. For example, if the browser language or Globalization element's User Culture attribute is set to `English/United Kingdom`

`[en-GB]` then setting a Format attribute to *Currency* will result in numeric values being shown with the Pound symbol (£). For more information about individual formatting options, see *Format Data*.

Customizing Dates and Calendars

A number of individual elements allow the user to enter dates or interact with calendars, and developers may want to change the behavior of related elements based on culture settings. These include the Input Date, Date Picker, Data Calendar, and Time Period Column. The format and presentation of calendars and dates varies widely depending on country and/or language. At the application level, date-related aspects of your Logi application can be changed by adding and configuring a Globalization element in the application's _Settings definition. For example, setting the Globalization element's **First Day of Week** attribute will change the display of calendars. If the attribute value is set to *1*, then *Monday* will be the first day in the calendar; the default is *0* which is *Sunday*. In addition, the browser's preferred language setting will affect the display of the month and day-of-week names in both a Logi element calendar and the selected value:



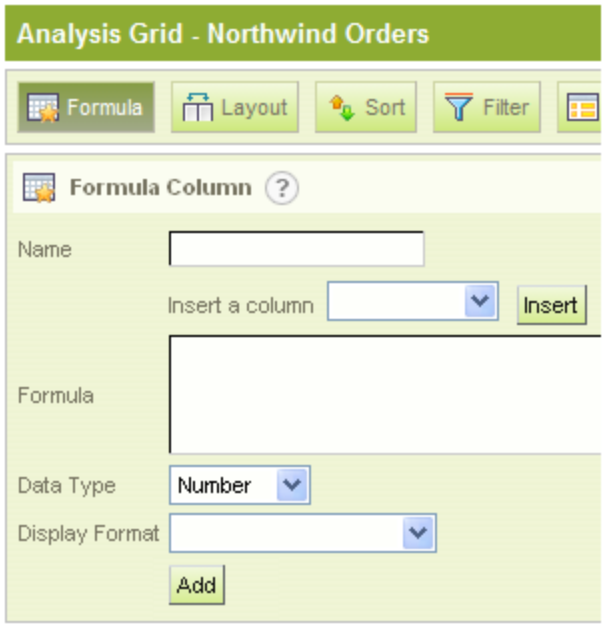
In the examples shown above, a Globalization element has been used to change the first day of the week to Monday. Notice that the **month name** and day-of-week **abbreviations** in the calendar change based on the browser's preferred language setting. In addition, the actual value returned after the user clicks on a date, shown below the calendars, also reflects the language pref-

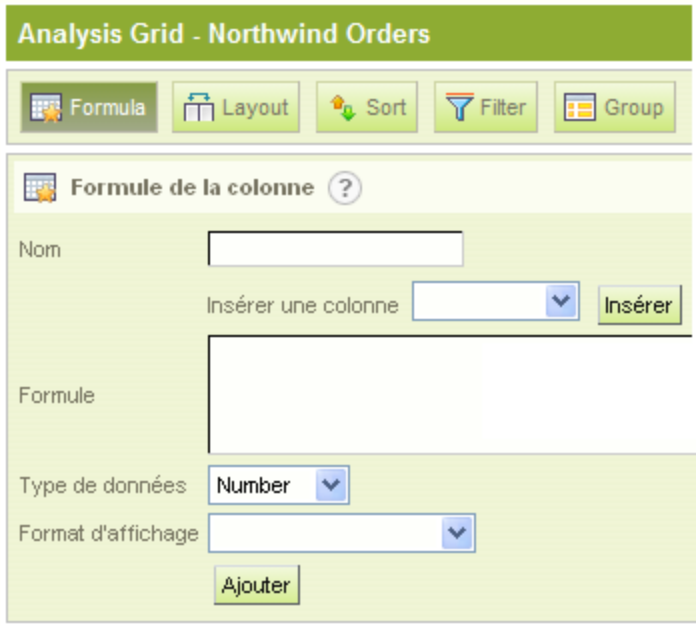
erence in any text it contains - "Jan" vs. "janv". The element's Format attribute, of course, controls the format of the date value returned into the element's input text box after the mouse click.

 In a "globalized date" scenario, when passing/submitting dates from your input controls to other definitions for use within tokens, as filters on queries, or within elements such as Condition Filters, we *highly* recommend utilizing the ISO-standard date format (yyyy-MM-dd) to avoid any locale/browser date conflicts that might arise. The element's **Input Date Reformat** attribute can be used for this purpose.

Customizing Super-Element Interfaces

Super-elements, such as the **Analysis Grid**, include their own user interface (UI), which allows users to interact with the element at runtime. By default, the labels, selections, titles, etc. displayed in the UI are in English. However, these can be modified to be displayed in alternate languages, using a Logi technology called Template Modifier Files.





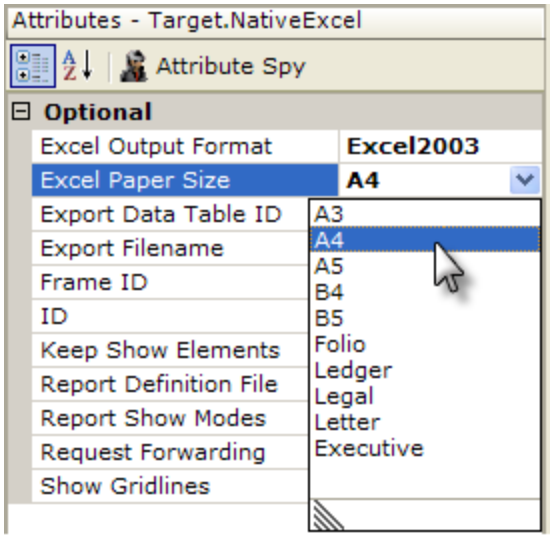
This technology allows the text used in the UI to be changed on-the-fly when the report pages are rendered at the web server. It can even be conditional, through the use of tokens, so that the changes are made automatically, based on external criteria such as the browser culture. This subject is discussed in detail in *Template Modifier Files*.

Customizing Report Defitions

Logi Info includes a technology called Definition Modifier Files (DMF). A DMF is an XML file that contains instructions to modify a definition file's elements and their attributes at runtime. It's a separate file that's processed *before* the Logi Server Engine starts to render a report definition. At that time, elements may be conditionally inserted or removed, and attributes may be set or unset. This makes it possible to set culture- or customer-specific values based, for example, on locale or security roles, at run-time. This means that each user can potentially receive a different report because the definition used to generate their report has been customized for them, on the fly. DMFs are similar to the Template Modifier Files, mentioned in "Customizing Super-Element Interfaces " on the previous page, and they're also conceptually similar to a plug-in that executes on the "LoadDefinition" event. In this context, you might think of them as the "poor man's plug-in": they can provide similar functionality but don't require additional development tools (Visual Studio, etc.) to write and compile a plug-in .dll or .jar file. To assist with globalization, a DMF could be used, for example, to perform language translation. XPath notation can be used in a DMF to find and replace specific strings of text anywhere in a report definition, in order to translate them into another language on-the-fly. You could even have a different DMF for each language supported and tokenize the DMF name in your report definition so that you can support multiple languages automatically. This subject is discussed in detail in *Template Modifier Files*.

Configuring Paper Size

Logi reports are most often viewed on a computer monitor but they can be printed to paper. Standard paper sizes vary in different countries and so Logi developers have been given some control over paper size. The following elements are involved (click the link to view more detailed information about their usage): The Printable Paging element allows developers to set the paper size and margins on a page to be printed by specifying the exact sizes in inches. For more information about the Printable Paging element, see Creating Printable Reports.



When exporting to Excel, the Target.Native Excel element includes a special attribute, **Excel Paper Size**, shown above, that sets the default paper size for the resulting worksheet. If left blank, the default paper size of the printer is used. For more information on the Target.Native Excel element, see *Export To Native Excel*.

Logi Studio Troubleshooting Guide

This troubleshooting guide is provided for developers who are new to the Logi Studio environment.

The following topics discuss the most common errors encountered when starting to develop Logi applications, and their solutions:

Installation and Setup Issues

The following issues should be considered if you have problems installing or upgrading your Logi products:

- **The installation behaves strangely**
  When installing Logi Analytics products, you must be logged into your computer as an Administrator or use the "Run As Administrator" option when executing the installation file. Otherwise, files and folders may not be installed correctly. If the product was installed using a non-administrative account, login as an Administrator, re-run the Logi installation file (right-click then "Run As Administrator") and uninstall your Logi product, and then reinstall it using the same technique.
- **Application can't be found on the web server**
  Within Logi Studio, "registering" a Logi application on the web server creates a "web application" or "virtual directory". Some user accounts may not have the appropriate permissions to do this. That may be the case if you attempt to register your application and nothing happens at the server. There are two alternate ways you can register your application and both require that you login to the web server: 1) launch the Logi Server Manager tool on the server and then use the "Add an Application" button to register it; or 2) launch the web server's management application and manually create a new virtual directory for your application.
- **Error: "You are not authorized to view this page"**
  If you receive this error while browsing a report application, make sure "Default.aspx" appears in the list of default documents in the virtual directory for your application. Also ensure that file system permissions are configured properly.
- **Can't run Logi applications on my development computer**
  Your Logi applications must be registered with the web server and (Windows app) .NET 4.x, or the (Java app) Oracle JDK or OpenJDK 8, 11, 12, 13, 14 must be installed to run reports locally. Use the Server Manager tool or run the registration wizard. Ensure that *localhost* is specified as the server name.

   Oracle has changed its Java usage policies -see *Java Usage Policy*for important information.
- **Applications won't run on my Apache-Tomcat server**
  Logi applications can be built to use either .NET or Java runtime libraries; be sure that you have used the correct version for

your server OS. On a Windows platform, .NET applications are intended to run on a web server that supports ASP.NET extensions so, generally, you must use Microsoft's Internet Information Server (IIS) or Cassini web server. On both Windows and Linux/UNIX platforms, Logi Java applications can be used with a variety of Java-based web servers; see *System Requirements - Logi Info*for more information. Logi Java applications will not run on Apache by itself; the Tomcat container is needed to run required servers.

- **How do I upgrade Logi applications?**
  You must install the new version of Logi Info, which will not overwrite any files you've edited, by running its installation program. Then, you must also change the version of each of Logi Info application you've created, using tools found in Logi Studio. When you open an application in Logi Studio, you will find a **Change Version** link in its Application tab that allows you to upgrade the current application. To upgrade multiple applications simultaneously, use a utility called Logi Server Manager (see the Studio Tools menu). For more information about upgrading products and changing versions see *Logi Product Upgrades*.

Debugging an Application

Logi Studio also provides **debugging** features for an application, but they have to be enabled to work. They are *not* enabled by default. You can turn on debugging for all your definitions by clicking the **Debug** icon in Studio's main menu and selecting *DebuggerLinks* from the selection list.

Clicking the icon sets an attribute value in *The _Settings Definition*, General element, and you can also turn debugging on by setting that value directly.

When the application is run, each page will display a debug icon in the bottom-right corner of the browser window. The Debug Trace page this link leads to contains detailed information about the page execution, including account names, request variables, database query results, error messages, etc. and is a very good way to identify errors. Each chart on a page will also have its own debug icon and trace page.

More detailed information about Logi application debugging techniques can be found in *Debug Reports*.

Development Problems

This topic introduces the different types of development problems you could run into when developing Logi Studio, as well as possible solutions.

Problems exporting to PDF, Excel, and Microsoft Word:

- **PDF** - Developers must give users the ability to export a report manually or instruct the server to automatically export a report. Manual exports are configured within Report definitions and automated exports are configured within Process definitions. If any elements in a definition have duplicate element IDs or their IDs are not set, they must be uniquely identified for the export to work.
- **Excel** - If error messages are received during the export process, remove any changes that have been applied recently to the report. If the error messages persist, the page may have been cached. You may need to terminate the current session and/or restart your web server to clear it.
- **Microsoft Word** - Ensure that the Target.Export element's Report Definition File attribute does not have a value of "Current Report." This value causes the Request parameters *not* to be passed to the export.

Problems Running Queries on the Web Server:

- **Application is not found**
  Using the management tool supplied with your web server, confirm that the application is registered with it, and that its virtual directory exists. If using .NET, ensure that the Application name exists and that the ASP.NET setting is correct in the virtual directory properties.
- **Cannot connect to the database**
  Native Connection elements (Connection.MySQL, Connection.Oracle, etc.) provide optimized connections to their databases. However, if you're using a connection with a Connection String that you entered manually, be sure it's valid, without

misspellings or missing information. Ensure that the password is being saved with the string. Information about connection strings for a variety of databases can be found online. If you must use Windows integrated authentication with the database, the ASP.NET account must be given access to the database or .NET impersonation must be used on the developed reports.

- **Query Builder or Database Browser connects to data OK but my report does not**
  This strange-seeming circumstance comes about because the Query Builder and Database Browser tools connect directly from your development PC to the datasource, whereas, when you preview or browser your application, your report is connecting through the web server and Logi Server Engine. When the Query Builder or the Database Browser connects but the report does not, the problem is often that the account used by the web server to run your Logi report (by default in .NET: the Application Pool account, ASPNET, or NETWORK service) does not have permission to access the datasource.

Problems Running the Wizards:

- **Wizards won't run**
  For the wizards to run properly, security should *not* be enabled in the _Settings definition for your reports. Before running the wizards, resolve any errors received while browsing or previewing the reports in a web browser. Web server-to-datasource connection configuration errors may prevent the wizards from running properly. Elements affected by wizards, such as "Add Data Table Columns", will not work correctly if the element is missing its required, unique element ID.

Problem Saving Logi Application:

- **Insufficient Disk Space**
  The user must have enough disk space to save files on the system. Files must not be set to "Read Only" in order for them to be overwritten; check their properties in the file system.

Problems Running the Debugger:

- **Debug link or icon doesn't work**
  Changes to the _Settings definition, like turning on debugging, may require a new browser session. Confirm that a new browser window is being launched to test this. The _Settings definition Debugger Style attribute value must be "DebuggerLinks"; the value is case-sensitive and should be one word.

Problems With Logon From Non-Standard Logon Form:

- The standard Logi logon page, rdLogon.aspx, passes the login name and password form variables and also a hidden form variable called "rdFormLogon" set equal to *True*. The latter variable should also be passed to the report if the username and password are being submitted from any other report definition or form.

Database Problems

This topic introduces the different types of database problems you could run into when developing Logi Studio, as well as possible solutions.

Problems Connecting to Database

- If using non-native connections (JDBC, ODBC, OLEDB, etc.) confirm that the Connection String has a valid DataSource value.
- Confirm that the database server is started and accessible via the network.
- Check that the password is being saved in the database connection string. This must be part of the connection string to ensure that a connection is established.
- If connecting to a database via OLEDB, make sure to use the appropriate provider. For SQL Server, this is the SQL Server Provider. If you have a choice of OLEDB or ODBC providers, we recommend using the OLEDB provider for all databases. For help creating an OLEDB connection string, use the "Connection String" wizard.
- If using a native Connection element, ensure that you're using the right one. For example, there is a Connection.MySql element for use with MySQL databases, Connection.SqlServer for MS SQL Server, etc.

Creating a Single Result Set From Two Databases:

- When JOINing tables, ensure that SELECT statements avoid ambiguous results by qualifying column names with table identifiers.
- A UNION of query results can be created by adding additional datalayers to elements like the Data Table element. This will combine the results into a single result set.

Datalayer Problems

This topic introduces the different types of datalayer problems you could run into when developing Logi Studio, as well as possible solutions.

Problems Displaying Query Results:

- To display query results, use the token @Data.column_name~ to access the data in each column returned. Token names are *case-sensitive*, so ensure that the case of the token matches the column name returned in the datalayer.
- Data returned into a datalayer can be examined using links on the Debugger Trace Report; turn on debugging links to view this page.

Joining Datalayers:

- Relationships between datalayers in a report can be created using a Join element, see *Join Datalayers*. This can be used to produce SQL-like joins of data from SQL and non-SQL datasources.

Logi Support

If your firm has a **Logi Support Plan**, you may occasionally work with one of our Support Engineers to resolve issues.

The following topics provide instructions for related activities:

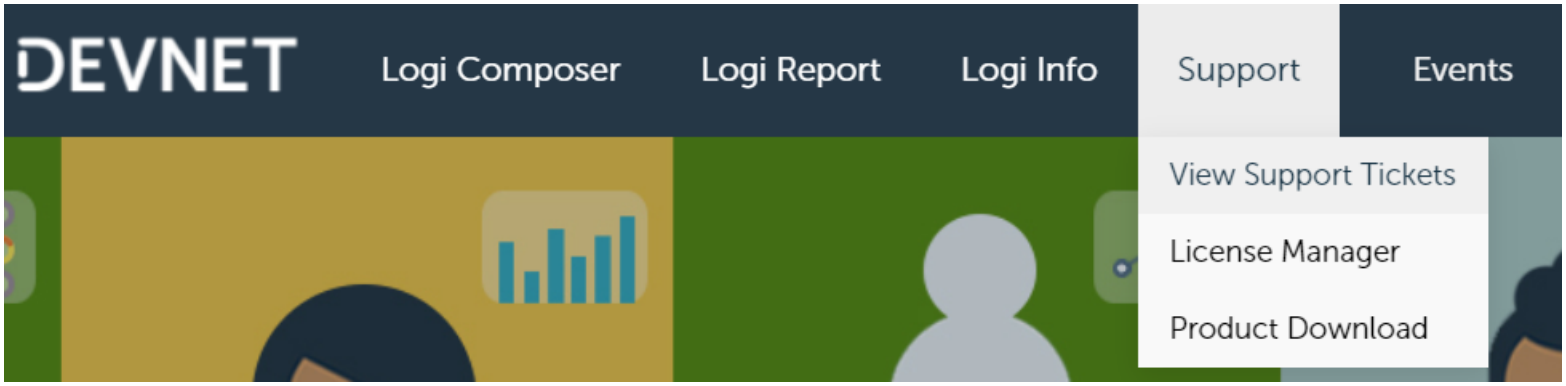
- Opening a Support Case
- Attaching Your Debugger Trace Report
- Attaching a Logi Info Definition
- Attaching a Complete Logi Info Application
- Attaching Discovery Logs and Settings Files
- Attaching Bookmark Files
- Preparing for a Webex Meeting

Opening a Support Case

On occasion, you may need help with your Logi product. If you need assistance with product *licenses*, contact Customer Service by email. Otherwise, to interact with a Logi Support Engineer, you need to open a **Support Case**.

Who can open a case? One or more of your colleagues is designated as the "Support Point-of-Contact (POC)" and only they can open cases. If you're not a Support POC, please get someone who is one to open a case on your behalf.

To open a Support Case:



1. Go to DevNet and login
2. On the menu, select **Support** → **View Support Tickets**, as shown above
3. In the portal, click **Open new case**

If you need assistance with your DevNet login credentials or can't access the Support Portal, contact Customer Service by email.

Your DevNet and Logi Support Portal sessions are set to expire after two hours of inactivity; if you've left the portal open in your browser for longer than that, you'll be asked to login again. We recommend that you exit the portal by browsing to a different DevNet page as soon as you're done working it in.

A Support Case provides you with several benefits, including a response within an agreed-upon time frame, an audit trail of interactions with a Support Engineer, and timely progress notifications. You use the Support Portal to manage and track the progress of your cases.

What Happens Next

If your case is not a request for information, once it's assigned to a Support Engineer, he or she will review the information you provided.

 It's very important that you provide accurate product, version, system, etc. information when opening the case.

The Support Engineer will respond to you and, based on the case information, may request additional information. He or she will review the situation and try to reproduce the issue locally. Then they'll proceed through a series of logical steps (which vary by issue) to determine the cause and provide a resolution.

You can help greatly by providing a clear description of the problem. If there's an error message, please copy-and-paste it into the description. If the error message is longer than a few lines, please paste the inner-most error message into the case description and *attach* the full error message or stack trace to the case.

The following sections provide instructions for preparing and attaching to your case the kinds of information the engineer may request, and for participating in a Webex meeting.

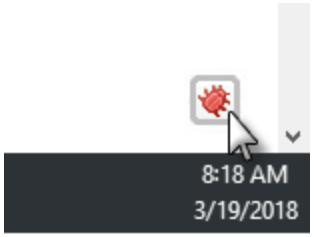
Attaching Your Debugger Trace Report

The **Debugger Trace Report** provides detailed information about the execution of your Logi Info application and you can learn about this feature in *Debug Reports*. A Logi Support Engineer may ask you to attach it to your case.

- Do not attach a *screenshot* of your debugger report page. It's easy to do but doesn't really help the engineer. Once your application is configured for debugging, use these steps to compress the debugger information and attach it to the case.
- The case attachment size limit is **25MB**. You may need to temporarily reduce the size of your data, through filtering or by setting the datalayer Maximum Rows attribute, if you believe the data, plus your report and debug pages (which are relatively small in any case), will exceed this limit.

Logi Info v12.5+ : Gathering Debug Information

With debugging turned on, you can view your report's (or chart's) debugger report page by clicking the debug icon:



For a report, the icon appears in the lower right-hand corner of the browser page. Each chart will have its *own* debug icon next to it and, if relevant, must be included individually.

You should also be aware that some operations, such as AJAX refreshes, scripting, or concurrently operating datalayer threads in a Dashboard, will generate their own secondary debugger reports.

Report	Paging Method	NoPaging
File Cleanup	Queued cleanup thread.	
	rdTempFileCleanupInterval	10 minutes
	rdTempFileLifespan	30 minutes
Debug	Completed	2018/03/19 1:27:10.0 PM
RefreshElement	Debug Information	View Debug Page

Links to these secondary pages look like the example shown above. Be sure you're attaching the correct debugger report page!

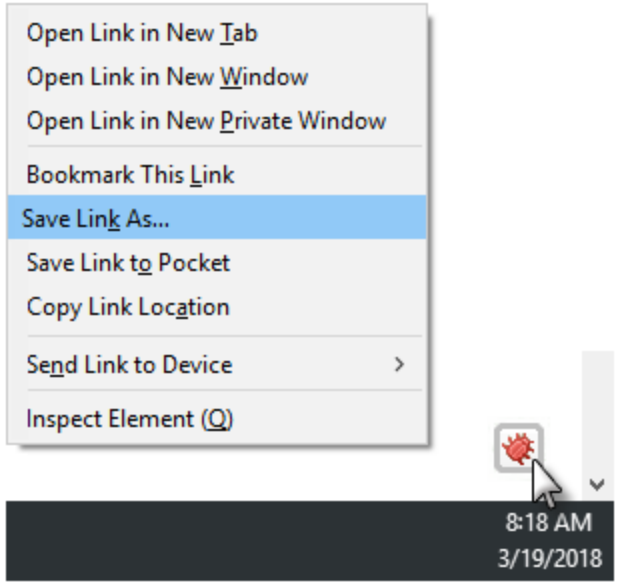
Report	Paging Method
Debug	Completed

* Generating this debugger information increases the overall elapsed time.

Once the debugger report is displayed, the link shown above appears in the bottom left-hand corner of the page. Click it to automatically gather, compress, and save the debug details to a .zip file. The suggested file name will include a really long random string; please rename it to something short and meaningful before saving it. Skip the next sub-section of instructions and go to the "Attaching the .Zip File" section.

Earlier Logi Info Versions: Gathering Debug Information

Earlier Info versions do not have a special link on the debugger report for compressing and downloading debug details, so you'll have to manually gather the relevant files and compress them into a file. Create an empty folder for collecting the files, then use these steps:



1. For a report, the debug icon appears in the lower right-hand corner of the browser page. Each chart will have its *own* debug icon and, if relevant, must be included individually. *Right-click* the debug icon, as shown above, select the *Save Link As...* or *Save Target As...* option (varies by browser) and save the file to your empty folder.

Get DataLayer.XMLFile	ID = dlOrdersXML	
	XMLFile Physical Filename	C:\inetpub\wwwroot\V12Test_SupportFiles\NW_Orders.xml
	Processed Row Count	10
	Processed Data	View File Stream Data (3,508 bytes)
	Finalize DataLayers	
	Data Engine	Data processing completed.
Generated DataLayers	File stream	View Data
Load XSL	Success	True

2. *Left-click* the icon to open the debugger report page and find the *View Data* link for the relevant data, as shown above.

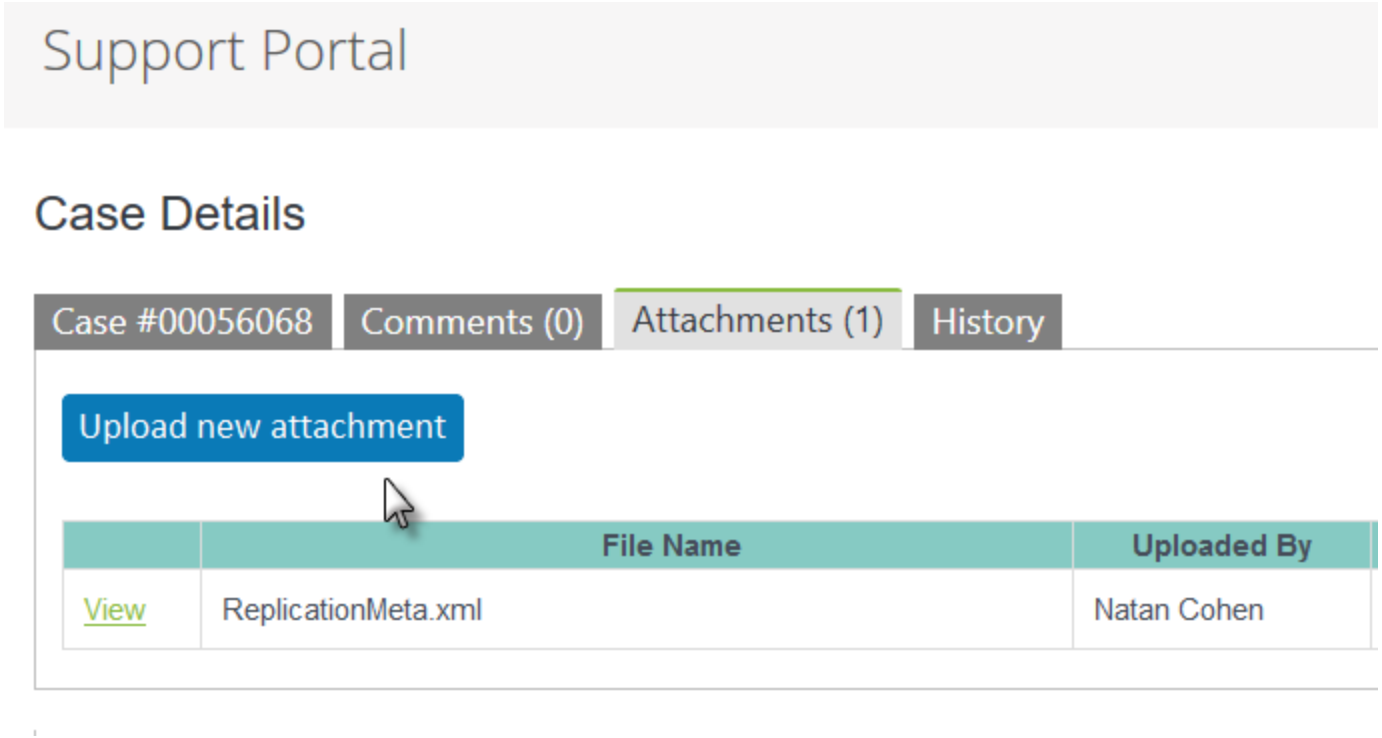
Depending on your Logi Info version, when you click the *View Data* link you may see a formatted table of data, as shown above. Right-click the *View raw XML* link and select *Save Link As...* or *Save Target As...* (varies by browser) and save the data file to your folder.

If you don't see the formatted table, then use your browser's Back arrow to go back to the debugger report, *right-click* the *View Data* link, and use the same technique to save the data file to your folder.

3. Repeat Step 2 as necessary to collect all of the relevant data.
4. Use the OS tools or a 3rd-party tool like WinZip to compress the folder and its contents into a single .zip file.

Attaching the .Zip File

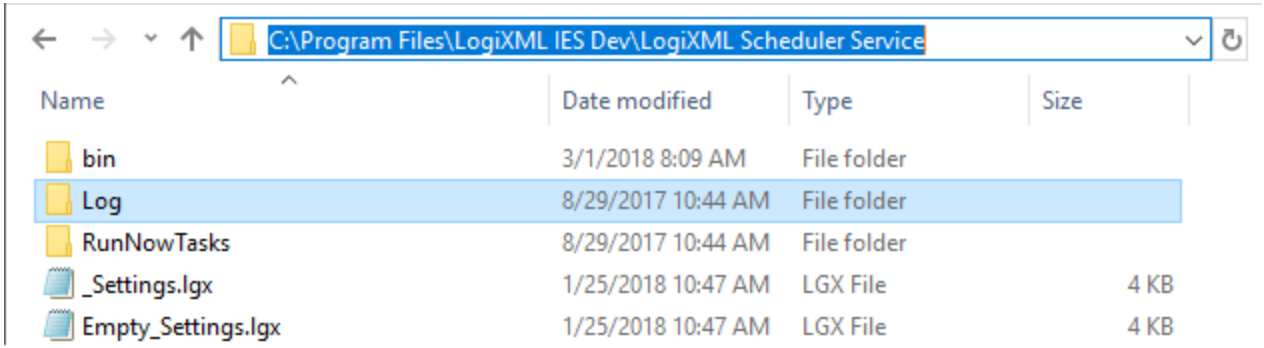
Once you've produced your .zip file, you're ready to attach it to the case. Go to the Support Portal, locate the case and browse its details.



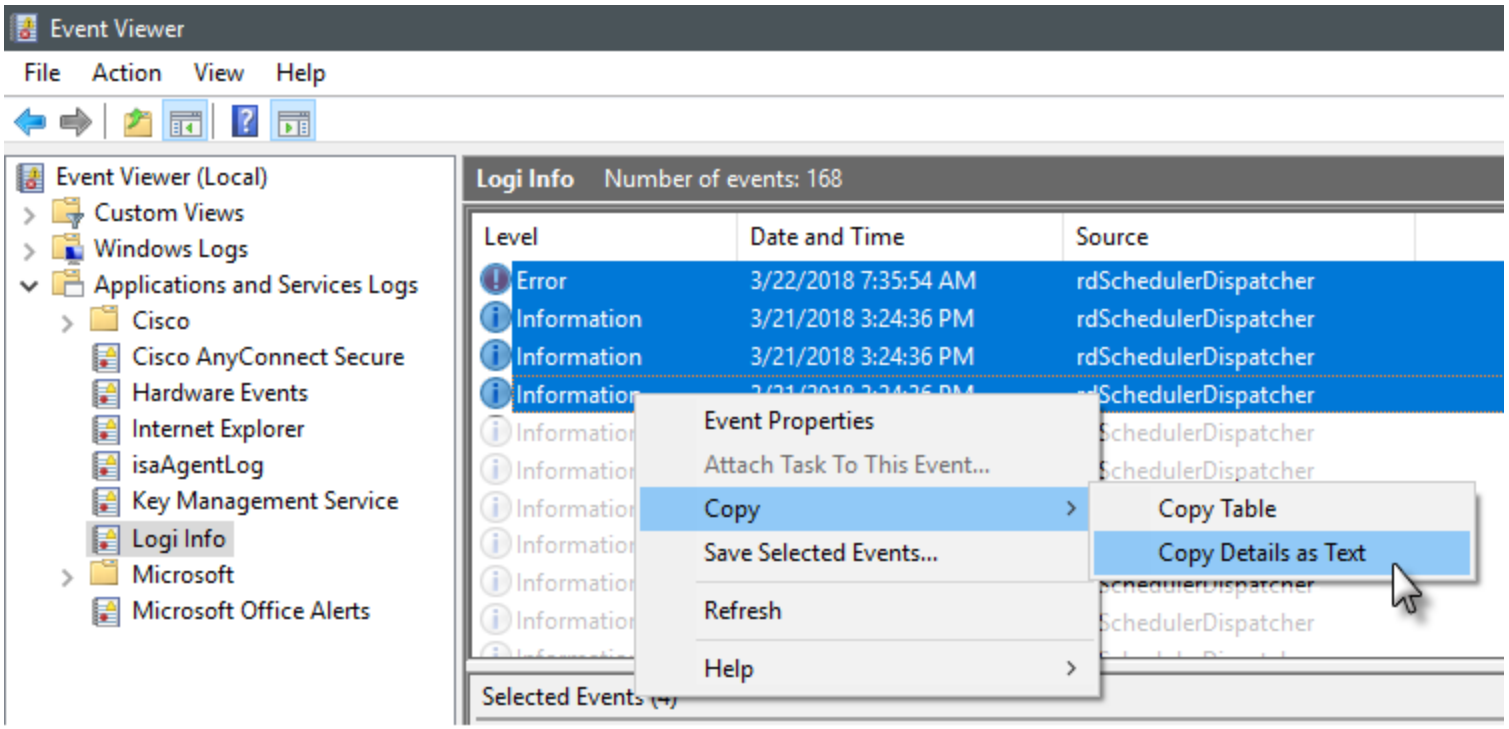
Click the *Attachments* tab, as shown above, and click *Upload new attachment* to select and upload your .zip file. Once uploaded, it will appear in the list of attachments.

Attaching Scheduler Debug Logs

The Logi Scheduler is a background service that can be used to run scheduled events. It logs its activities and can be configured to include several levels of information in its log, including very detailed debug details. See the **Scheduler's Settings File** section of *Logi Scheduler* to learn how to set logging to include debug information.



The Logi Scheduler Service for .NET logs operational information to .xml files in (if default installation location was used): `C:\Program Files\LogiXML IES Dev\LogiXML Scheduler Service\Log` as shown above. Find the log file for the relevant time period and attach it to your case, or copy it into a folder with other debug information to be compressed and attached.



In addition, the Scheduler will log events in the Windows Event Log, which can be seen in the **Event Viewer** tool. You can select one or more events and copy them, with their details, by right-clicking them as shown above. Paste the copied text into a text file and include it with your debug and log files. The Logi Scheduler Service for Java logs operational information to `<installFolder->/Log/LogiScheduler.log`, where *installFolder* is the folder chosen when the Scheduler was deployed. Relevant log files can be attached to the case or copied into a folder with other debug information to be compressed and attached. Attach the files to your support case in the Support Portal using the same technique described earlier.

Attaching a Logi Info Definition

Your Support Engineer may ask you to attach a Logi Info Report or Process definition file to your case. The .lgx file can be attached using the same technique described earlier. Here are some things to consider before doing so:

- "Sanitize" the attributes - If your firm is sensitive to such issues, you may wish to remove or alter database login or other credentials in the definition that belong to your customers. Generally, our legal support plan contract with *you* ensures that we'll handle all materials shared in this way confidentially.
- "Shrink" the definition - If a definition is long and complex, to make diagnosis easier, make a copy of it, and in the copy remove elements that have no bearing on the issue from it. For example, remove page sections (header, footer, etc.) that work fine and aren't related to or dependent upon the issue. Remove explanatory or "change log" comments. Remove features like Export elements if they're working correctly. Once you've trimmed down the (copied) definition, run it again to make sure the problem still exists. Then attach it to the case.
- Include dependencies - If Shared Elements, the _Settings definition, or a Master Report are necessary to run the report, be sure to include them, too.

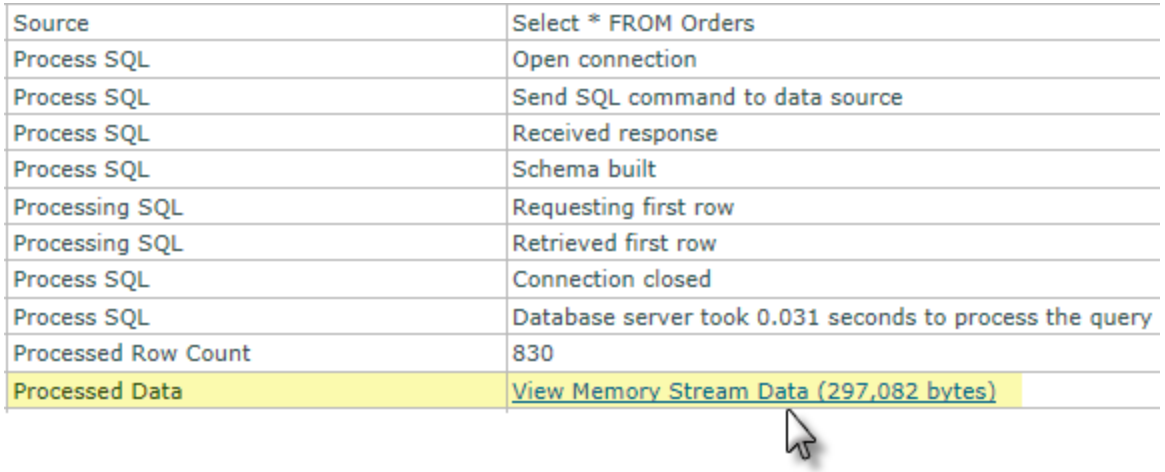
The more you can focus in on the elements that matter, the easier it will be for a Support Engineer to help you.

Attaching a Complete Logi Info Application

Your Support Engineer may ask you to attach your entire Logi Info application to your case. The application's files can be compressed and attached using the same techniques described earlier. Here's how to prepare your application before doing so:

The Sample Applications on DevNet include sample data in XML files, so connection details aren't necessary to run them. Assuming the connection is not part of the problem you need help with, you can do the same thing with your application's data using this technique:

1. Run the application and open its debugger report page.



Source	Select * FROM Orders
Process SQL	Open connection
Process SQL	Send SQL command to data source
Process SQL	Received response
Process SQL	Schema built
Processing SQL	Requesting first row
Processing SQL	Retrieved first row
Process SQL	Connection closed
Process SQL	Database server took 0.031 seconds to process the query
Processed Row Count	830
Processed Data	View Memory Stream Data (297,082 bytes)

   Find and click the link associated with the data returned from the source, before any other processing occurs, as shown above.
2. When the Data View page is displayed, click the *View raw XML* data link. The formatted XML data will appear in a new page.
3. *Right-click* anywhere in the XML data and select *View source* or *View page source* (varies by browser). The unformatted XML data will appear in a new page.

4. *Right-click* anywhere in the XML data and select *Save As...* to save the raw XML data in your application's _SupportFiles folder, with an .xml extension.
5. Repeat to create XML data files for all of the data required.

If your application depends on a "parent" application, "untether" it by modifying your application so that it can run stand-alone.

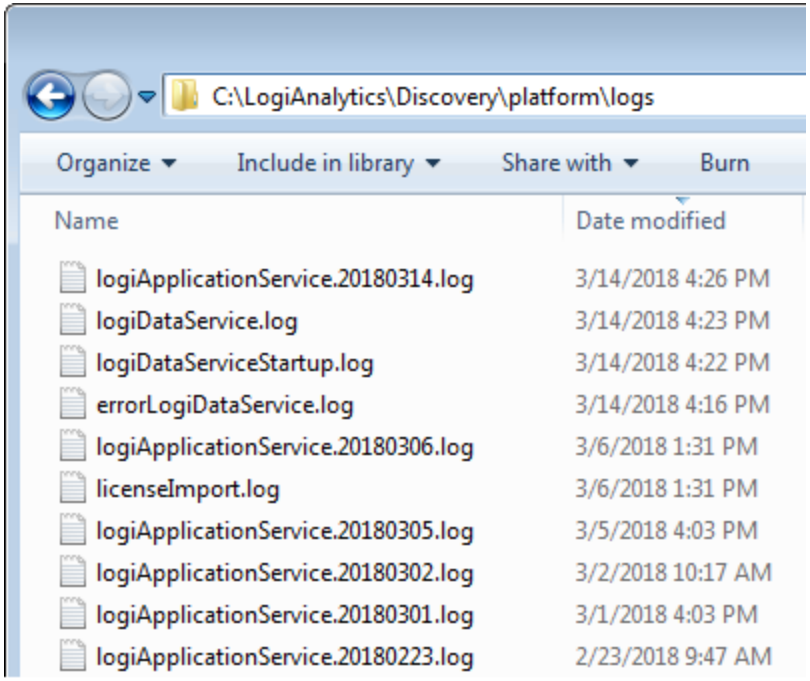
If your application implements security, be sure to provide a set of login credentials to us. If there are any critical session variables required, tell us about them, too.

Make a "groomed" copy of your application folder by including only the minimum files necessary. Do not include files like *newReport.lgx,* obsolete definitions, unused images and style sheets, etc. Remove the application's *bin*, *rdTemplate*, *rdDownload*, or *rdDataCache* folders.

Compress the groomed application folder and it's ready to be attached to the case.

Attaching Discovery Logs and Settings Files

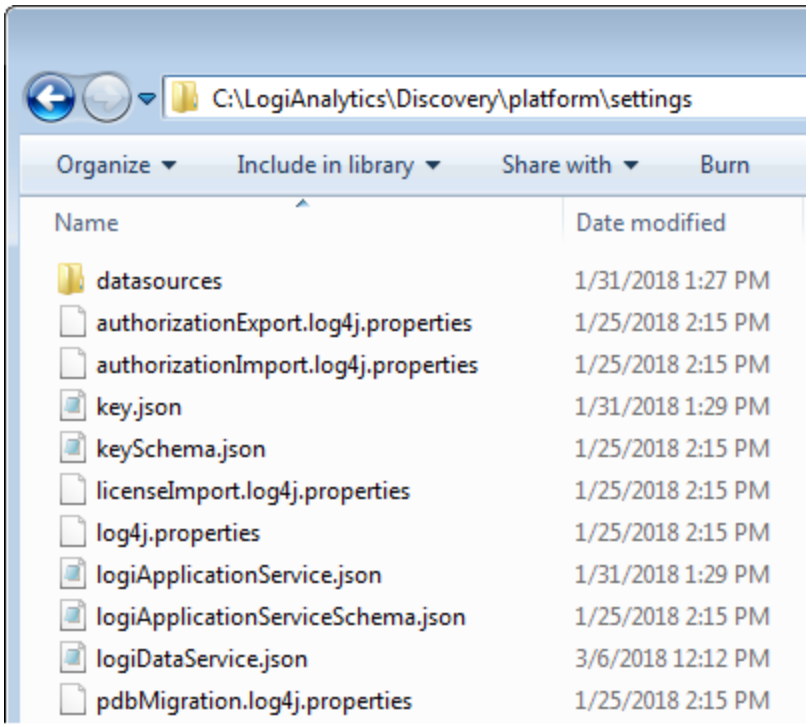
If you're using our **Discovery** add-on module, a Support Engineer may ask you to attach specific Discovery logs or settings files to your case. These files can be compressed and attached using the same techniques described earlier. Here's where to find these files:



General Discovery log files can be found in (assuming a default installation):

```
C:\LogiAnalytics\Discovery\platform\logs
```

as shown above. The Support Engineer will specify which files to attach.



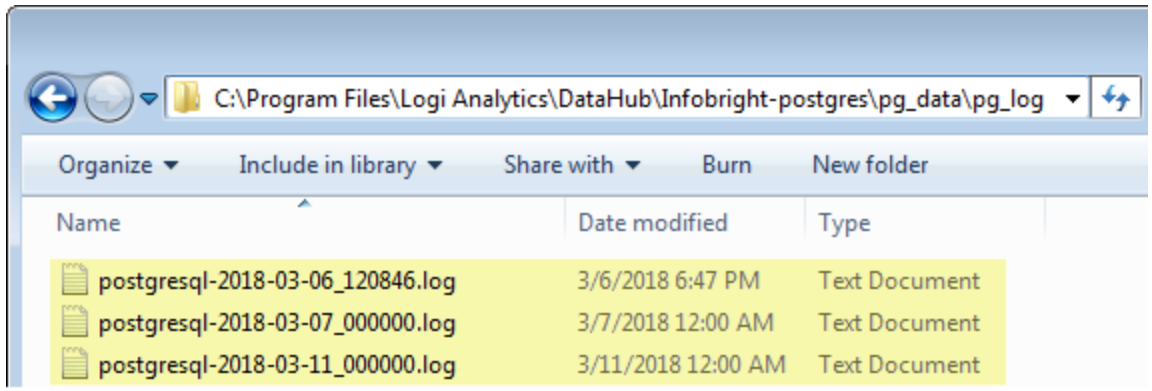
General Discovery settings files can be found in (assuming a default installation):

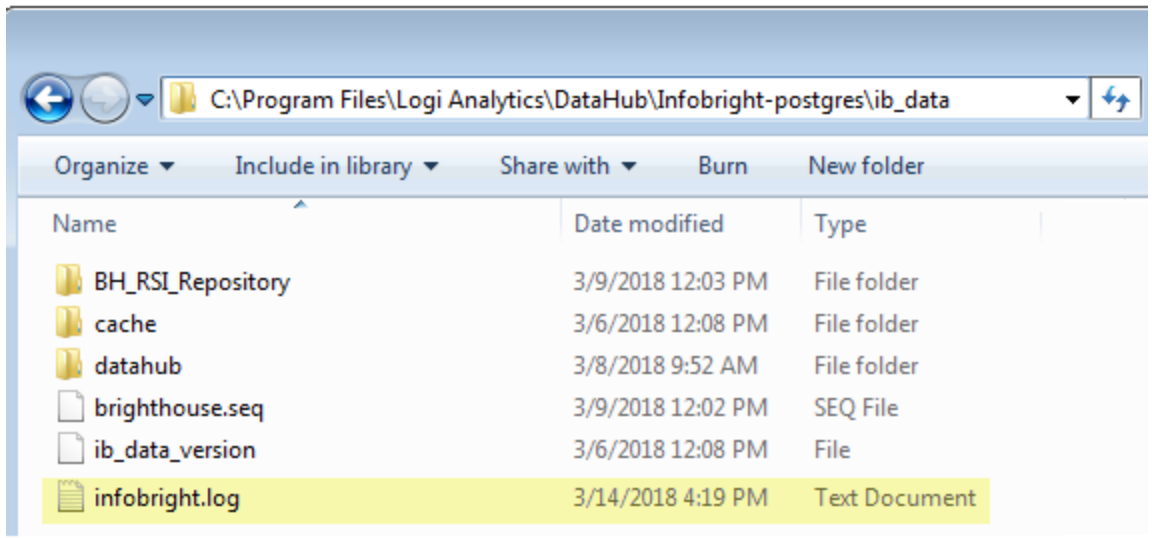
```
C:\LogiAnalytics\Discovery\platform\settings
```

as shown above. The Support Engineer will specify which files to attach.

Infobright-Postgres Logs

Discovery also works with our **DataHub** product, which uses an Infobright data repository, and a Support Engineer may want to see the Infobright-postgres logs.





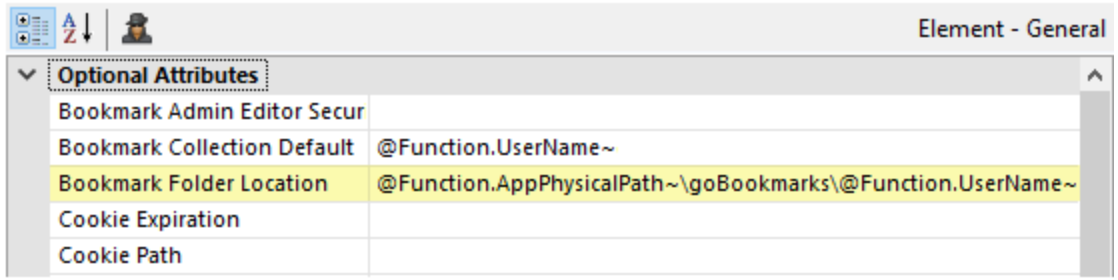
Infobright-postgres log files can be found in (assuming a default installation):

```
C:\Program Files\Logi Analytics\DataHub\Infobright-postgres\pg_data\pg_log
C:\Program Files\Logi Analytics\DataHub\Infobright-postgres\ib_data
```

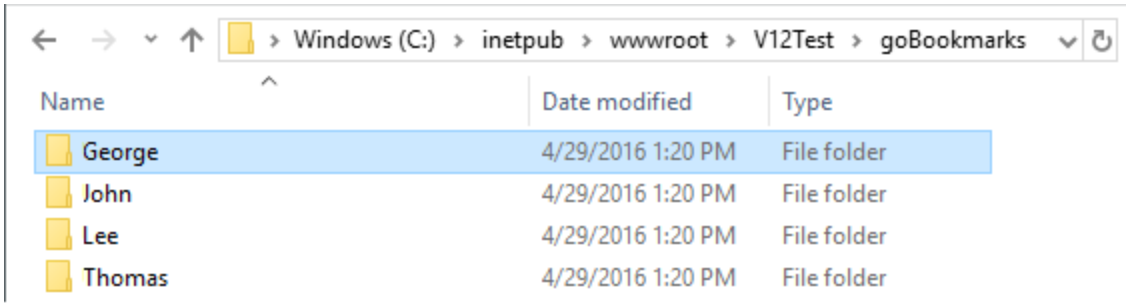
as shown above. The Support Engineer will specify which files to attach.

Attaching Bookmark Files

If you're using bookmarks, for example in our **Self-Service Reporting Module** add-on, the **InfoGo** application, or your own Logi Info application, a Support Engineer may ask you to attach Bookmark files (see *Bookmarks*) to your case. If you're not storing your bookmarks in a database, bookmarks are stored as .xml files, which can be compressed and attached using the same techniques described earlier. Here's where to find these files:



The location and naming of bookmark files is configured in your _Settings definition, **General** element, as shown above. Depending on your Logi Info version, you'll see a **Bookmark Folder Location**, or similarly named, attribute. In the file system, navigate to the specified folder:



Select one or more folders for compression and attachment.

Preparing for a Webex Meeting

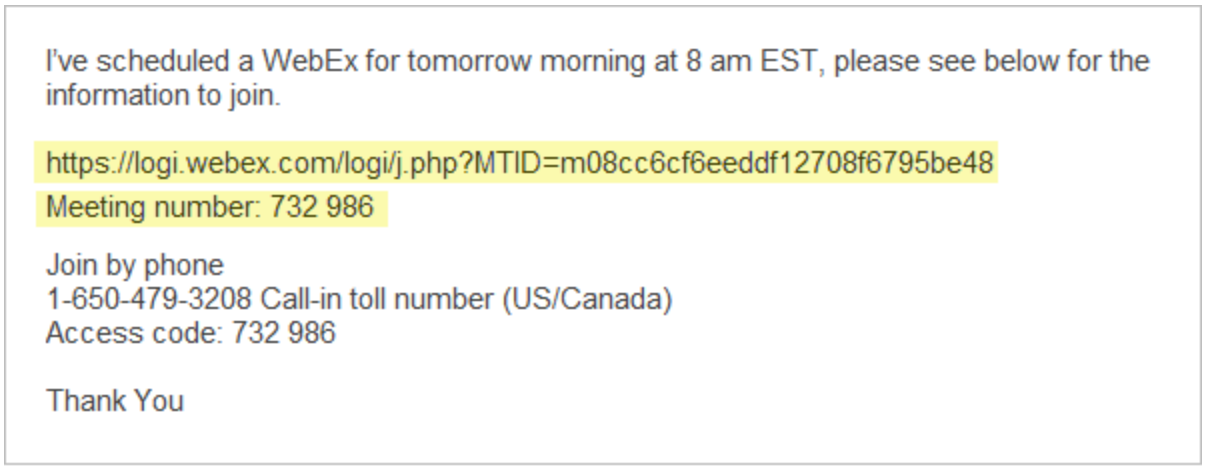
The Support Engineer may decide that a Webex meeting is the best way to see what's happening with your application. Cisco Webex is an online meeting and screen sharing service that allows you and the engineer to collaborate privately to diagnose and resolve issues. A number of computer configurations for sound and video are supported.

You can have multiple developers join the Webex meeting, but one of your Support POCs *must* be a participant.

To prepare for your Webex meeting:

- Make a backup of your application before the meeting, so that you can restore definitions, etc. after the meeting, if necessary.
- Start Logi Studio using the *Run As Administrator* option (right-click Studio icon).
- Be aware that, if Logi Studio is on a machine that you use Remote Desktop to access, during a Webex meeting the Support Engineer may be able to *see* Studio on that machine but may not be able to *interact* with it. You'll have to manipulate Studio, which will slow the diagnostic process down.
- Be ready to demonstrate the application issue.

The Support Engineer will set up the Webex meeting and enter the details in a Case comment:


I've scheduled a WebEx for tomorrow morning at 8 am EST, please see below for the information to join.

https://logi.webex.com/logi/j.php?MTID=m08cc6cf6eeddf12708f6795be48
Meeting number: 732 986

Join by phone
1-650-479-3208 Call-in toll number (US/Canada)
Access code: 732 986

Thank You

The Case comment will contain information similar to that shown above. Use the provided link to browse to the Webex site and then enter the Meeting number shown to access the meeting.

If you discover that you can't attend the meeting at the scheduled time, please enter a Case comment to let the Support Engineer know.

Cisco provides this online help for Joining a Webex Meeting, Connecting to Audio, and General Help.

File Access Permissions

A Logi Info application resides within an *application folder* on the web server.

The following topics discuss additional details for granting file access permissions:

About File Access Permissions

A Logi Info application, on your web server, consists of an *application folder* with a set of sub-folders and files. The application will need to write data into some of those sub-folders; for example, data caching, temporary file creation, and exports are a few of the operations that do that.

However, the **security settings** for these folders may prohibit file writing by accounts normally used to run web applications like a Logi application.

Applications created using the New Application wizard in Logi Studio generally set these file permissions for you on your development machine, as discussed below. However, if you manually create or copy an application, they may not be set correctly, so you may need to explicitly grant file access permissions to the application folders (which are identified below).

Tell-Tale Errors

If you receive these or similar errors when trying to preview or browse your report:

```
There was a problem running a DataLayer. The error was: The web application is unable to write to a folder.
```

`Unable to save to the data cache. The file was: <application file path>\rdDataCache\???.xml. Access to the path is denied.`

`Access to the path 'C:\inetpub\wwwroot\HelloWorld\MyData' is denied.` then it is most likely that you need to grant Write permissions to the *rdDataCache*, *rdDownload*, and/or other folders.

.NET Application? Let The New App Wizard Do It

If you use the New Application Wizard in Studio (File → New Application...) to create a Logi .NET application, it will set the appropriate file permissions for you. It actually provides a pretty wide-open security setting, granting all rights to everyone, and you may care to set more restrictive rights manually. You may, nonetheless, need to manually grant rights when you deploy your application to a production server.

Generally, if your application does anything more than put simple text on a page, granting file permissions will be required and you'll need to do this for each Logi application you develop and deploy.

Java Application? Usually Done for You

Which Accounts?

File access permissions should be assigned to these accounts:

Web Sserver OS	Grant File Access Permissions To
Windows	The Application Pool account (see below), or for older Windows Server systems that do not use application pools, the **Network Service** account.
Linux	The account that the web server uses to run web applications.

This presumes a default OS installation in which no special renaming or removal of default accounts has been undertaken.

If you're experiencing "Access denied to file XYZ"-type errors when running your Logi application, try temporarily giving *Full Control* permissions to **Everyone**. If the errors stop appearing, then you know for certain that file access permissions are the source of the errors.

Which Application Sub-Folders?

Your web server account will, of course, have to be able to *read* all of the files in your application folder.

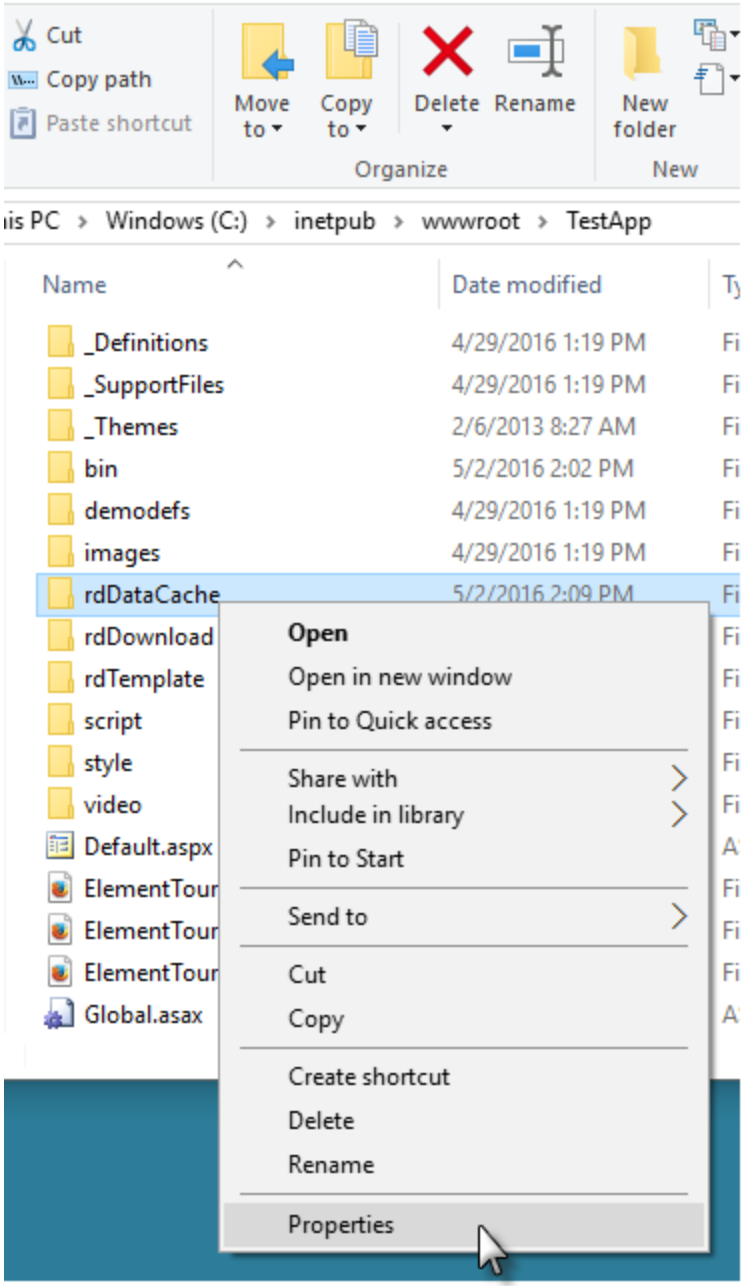
The two sub-folders in your Logi application folder that will routinely need to have *Write* permissions granted to them are **rdDataCache** and **rdDownload**. The latter is often used as a convenient place to hold temporary files and does not necessarily have anything to with actual downloads.

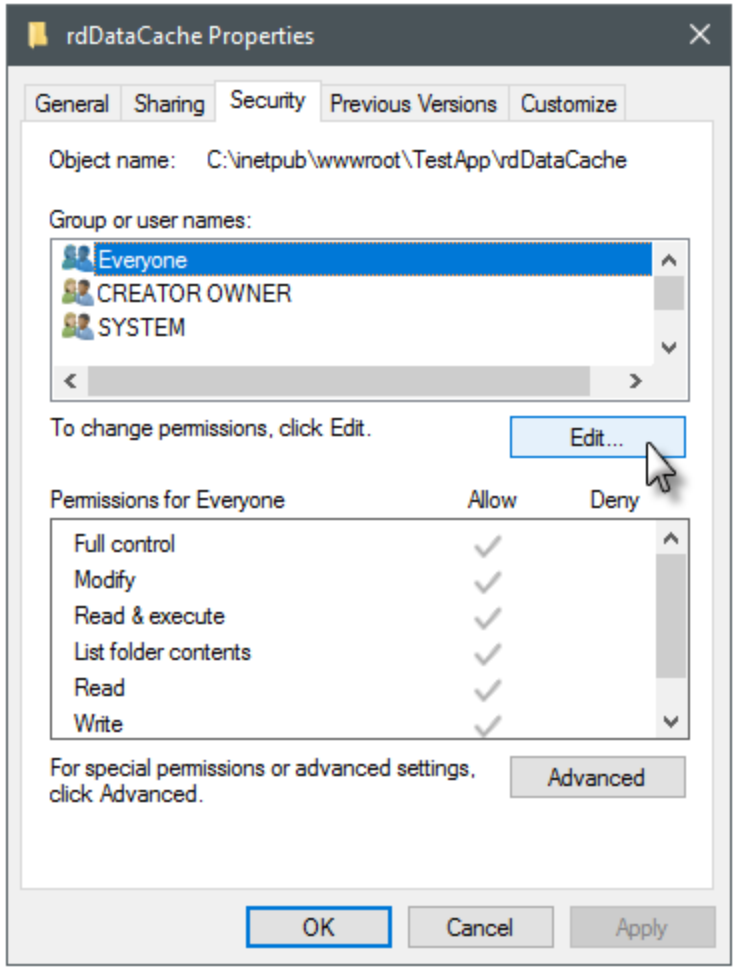
If, in the course of your development work, you add your own new sub-folders to hold data files or "save files" used with elements like a Dashboard, or decide to write data to another folder, you will need to grant *Write* and possibly even *Full Control* permissions to them as well.

One approach that guarantees access is to give *Full Control* permissions to the Logi application folder itself (which is what Studio's New Application wizard does). All files and folders within the app folder then inherit the permissions. However, this is not necessary and, as mentioned earlier, may cause security concerns.

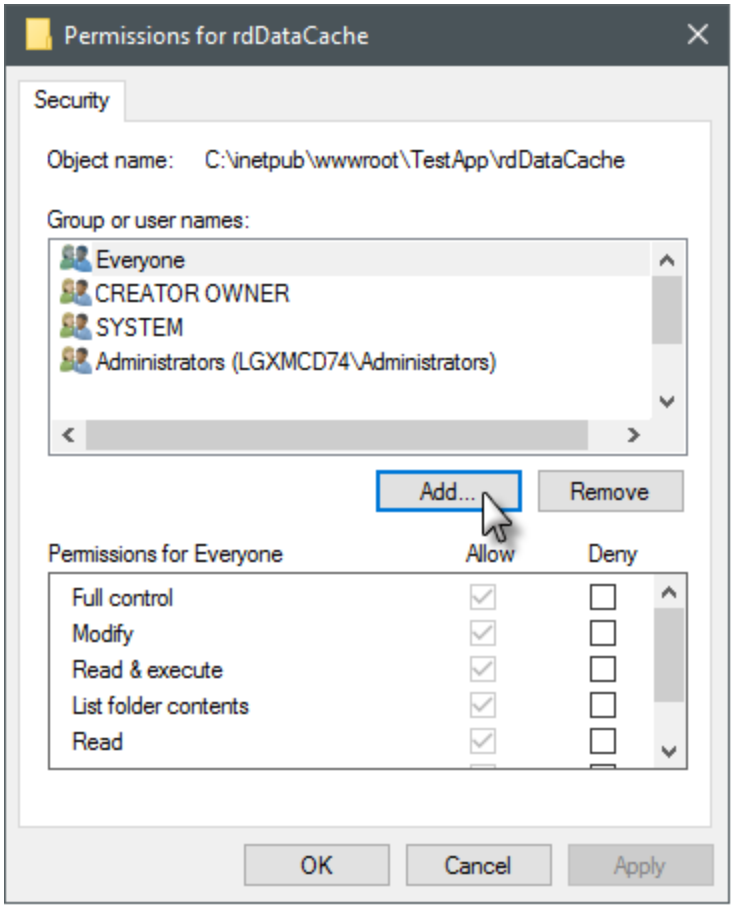
Granting File Permissions Under Windows

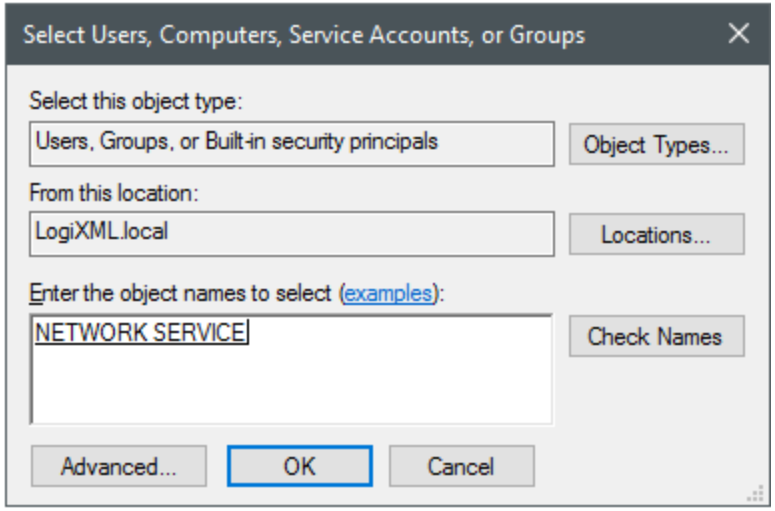
Depending on your OS, you'll use **File Explorer** or **My Computer** to set file permissions.


Cut
Copy path
Paste shortcut
Move to
Copy to
Delete
Rename
New folder
Organize
New
is PC › Windows (C:) › inetpub › wwwroot › TestApp
Name
Date modified
_Definitions 4/29/2016 1:19 PM
_SupportFiles 4/29/2016 1:19 PM
_Themes 2/6/2013 8:27 AM
bin 5/2/2016 2:02 PM
demodefs 4/29/2016 1:19 PM
images 4/29/2016 1:19 PM
rdDataCache 5/2/2016 2:09 PM
rdDownload
rdTemplate
script
style
video
Default.aspx
ElementTour
ElementTour
ElementTour
Global.asax
Open
Open in new window
Pin to Quick access
Share with
Include in library
Pin to Start
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties

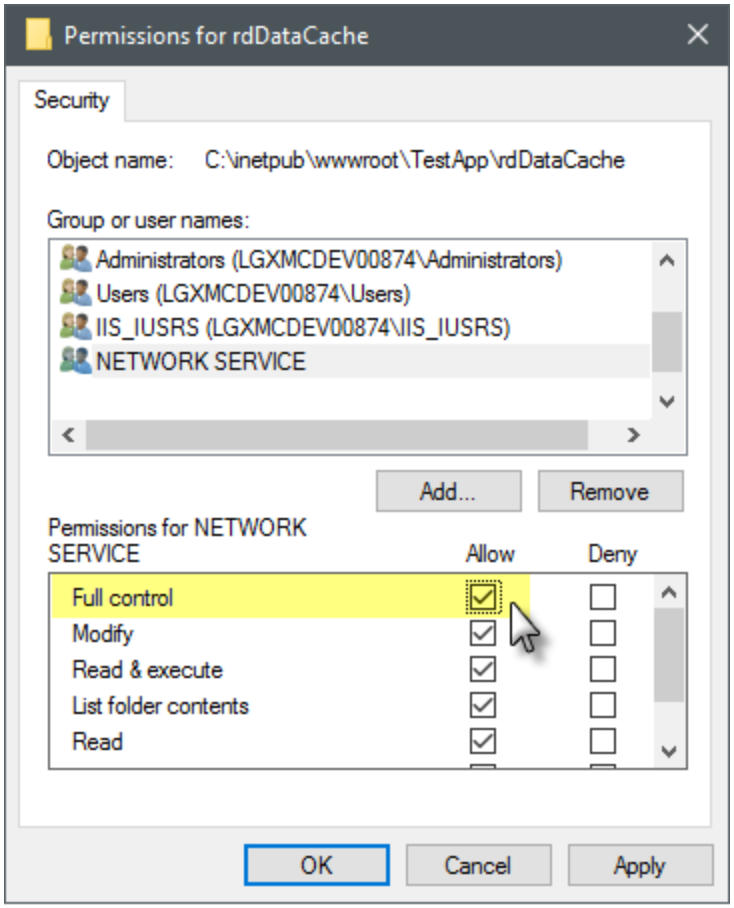

rdDataCache Properties
General
Sharing
Security
Previous Versions
Customize
Object name: C:\inetpub\wwwroot\TestApp\rdDataCache
Group or user names:
Everyone
CREATOR OWNER
SYSTEM
To change permissions, click Edit.
Edit...
Permissions for Everyone
Allow
Deny
Full control
Modify
Read & execute
List folder contents
Read
Write
For special permissions or advanced settings, click Advanced.
Advanced
OK
Cancel
Apply

1. On the web server, navigate into your Logi application's folder and select the *rdDataCache* folder.
2. Right-click the folder and select **Properties** from the pop-up menu, as shown above, left.
3. In the Properties dialog box, click the **Security** tab, as shown above, right. Click **Edit...** *No Security tab visible? See section below.*





4. If we're assigning permissions to the NETWORK SERVICE account, click Add... and then enter the account name in the subsequent dialog box, as shown above. Click **OK**.

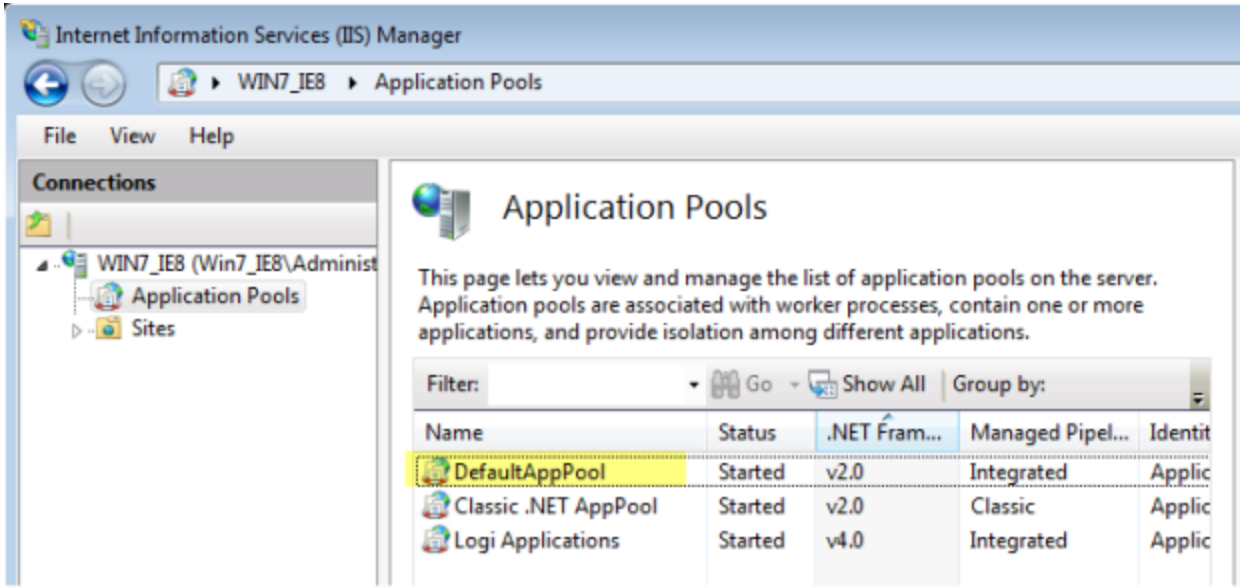


5. Back in the Security dialog box, ensure that the NETWORK SERVICE account is selected and click the *Full Control* check box, as shown above. Then click **OK**.

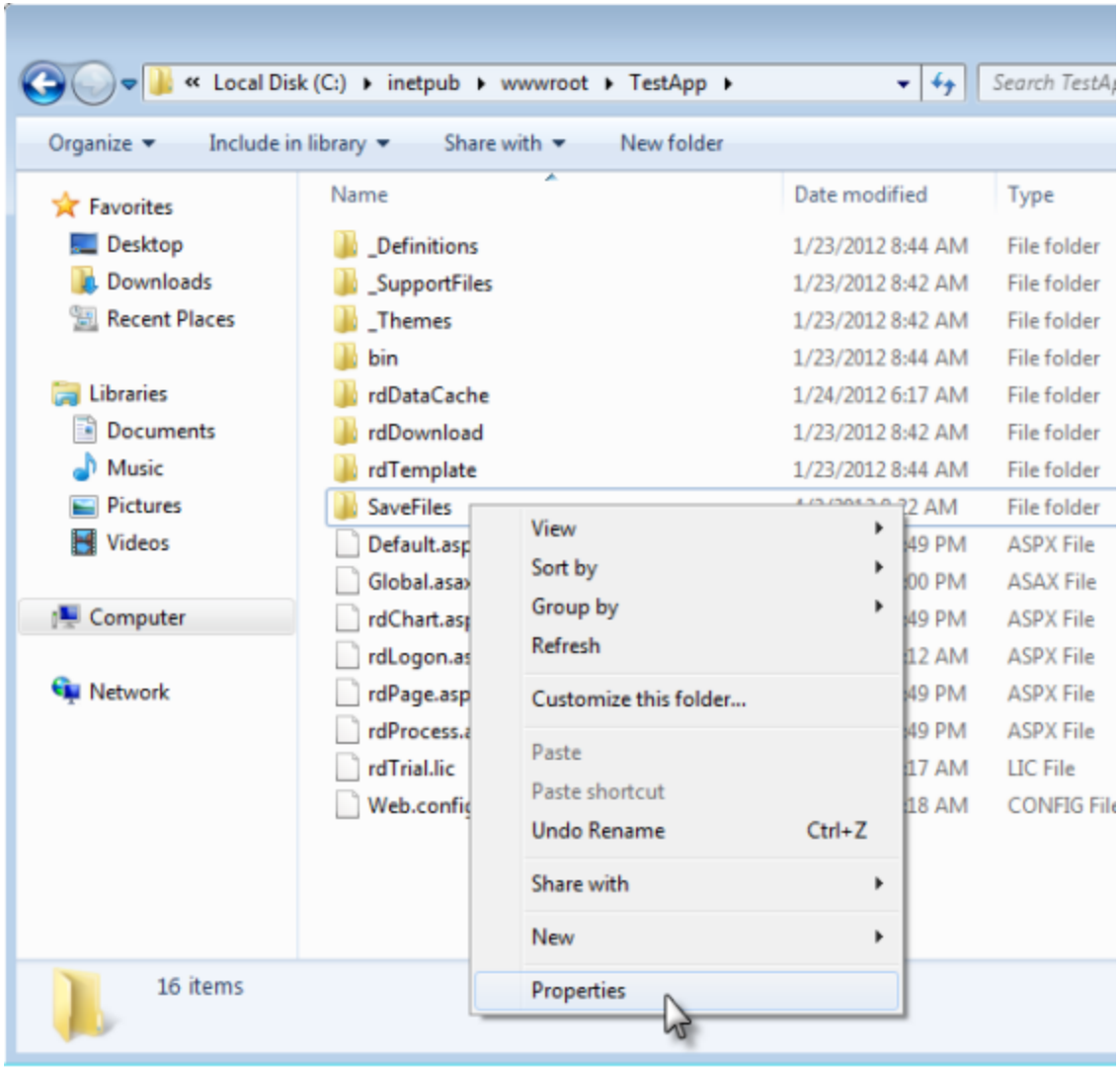
Repeat this process for the `rdDownload`folder and for any other folders that you'll be writing data to.

IIS 6 And Later - Application Pools

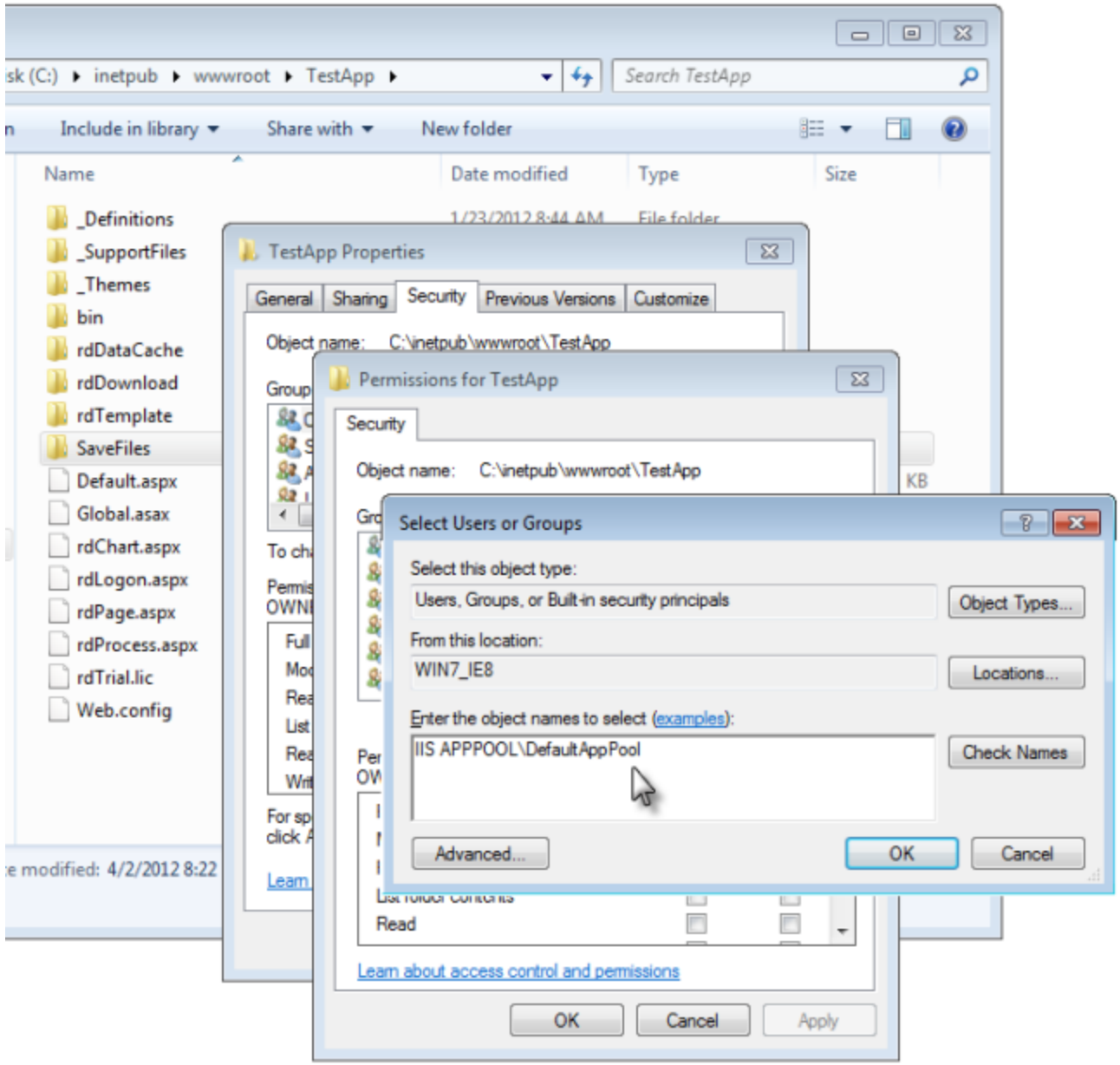
In IIS 6 and later, in addition to setting the permissions shown earlier, you may also need to give permissions to the "Application Pool". Logi Info applications are generally run by IIS within an application pool, which provides isolation and performance improvements for web apps that may use different versions of .NET. To assign file access permissions to an application pool:



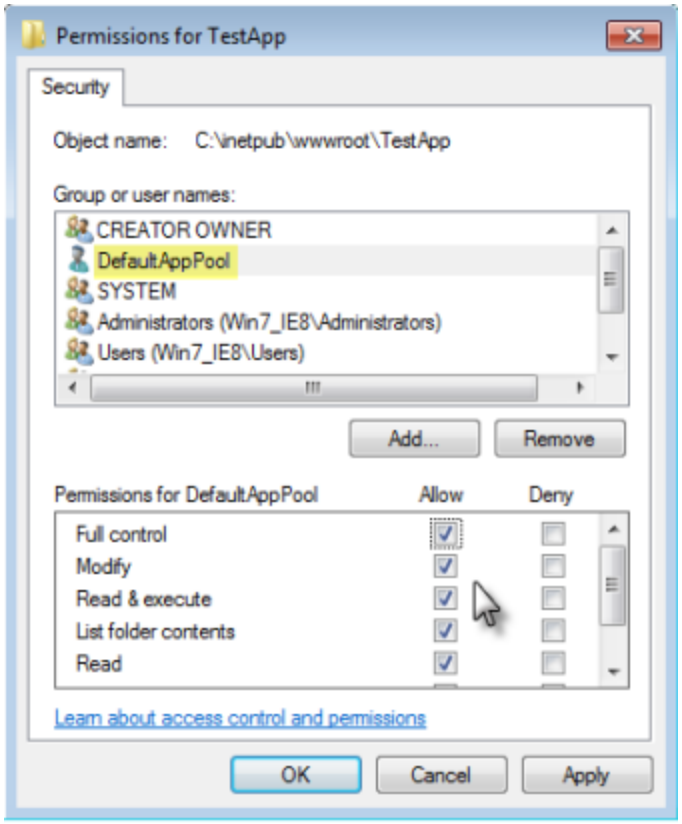
1. Open the IIS Manager utility, locate your Logi application and determine, by examining its properties, which application pool it's assigned to. Note the name of the pool ("DefaultAppPool" in the example).



2. In the example above, we're setting permissions for the sub-folder "SavedFiles". Use File Explorer to navigate to your Logi app folder, then select and right-click the sub-folder in question, and select **Properties**.


sk (C:) ▸ inetpub ▸ wwwroot ▸ TestApp ▸
Search TestApp
Include in library
Share with
New folder
Name
Date modified
Type
Size
_Definitions
_SupportFiles
_Themes
bin
rdDataCache
rdDownload
rdTemplate
SaveFiles
Default.aspx
Global.asax
rdChart.aspx
rdLogon.aspx
rdPage.aspx
rdProcess.aspx
rdTrial.lic
Web.config
TestApp Properties
General
Sharing
Security
Previous Versions
Customize
Object name: C:\inetpub\wwwroot\TestApp
Permissions for TestApp
Security
Object name: C:\inetpub\wwwroot\TestApp
Select Users or Groups
Select this object type:
Users, Groups, or Built-in security principals
Object Types...
From this location:
WIN7_IE8
Locations...
Enter the object names to select (examples):
IIS APPPOOL\DefaultAppPool
Check Names
Advanced...
OK
Cancel
Read
Learn about access control and permissions
OK
Cancel
Apply

3. In the Properties dialog box, click the **Security** tab, and then the **Edit** button.
4. In the Permissions dialog box, click **Add**.
5. In the Select Users or Groups dialog box, enter "IIS APPPOOL\<the app pool name>" as shown above.
6. Click **OK**, to close the dialog box and return to the Permissions dialog box.



Finally, select the application pool in the names list, and set its permissions, as shown above. Then click **OK**.

No Security Tab Visible?

In order to make the **Security** tab visible in the Properties dialog box, you must turn off **Simple File Sharing**. This is usually an issue only on stand-alone computers.

1. Open **Control Panel** and locate the **Folder Options** link. In Windows XP, this is through the **Appearance and Themes** link. Or, instead, you can access Folder Options through the **Tools** menu item in My Computer.
2. In the Folder Options dialog box, select the **View** tab.
3. At the very bottom of the list of **Advancedsettings**, un-check the "Use simple file sharing" option. Click **OK**.

The security tab should now appear when you select a file or folder's properties.

Windows IIS Configuration

When Logi Studio "registers" a new Logi application with the Internet Information Services (IIS) web server under Windows, it attempts to complete a number of configuration activities to ensure that the application will run correctly.

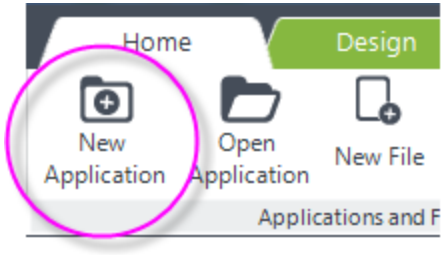
The following topics provide additional information describing those activities and instructions covering how to complete some of them manually:

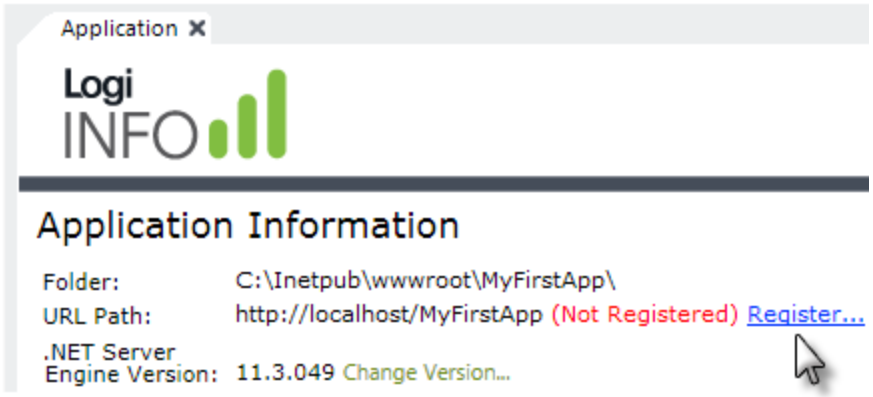
- Studio's Configuration Steps
- Starting the Default Web Site
- Manually Creating an IIS Web Application

Studio's Configuration Steps

Logi Studio attempts to register a Logi application with the IIS web server when:

- You create a new application using Studio's **New Application** wizard, or
- You click the **Register** link in Studio's Workspace Application tab, or
- You download a DevNet **Sample** application.

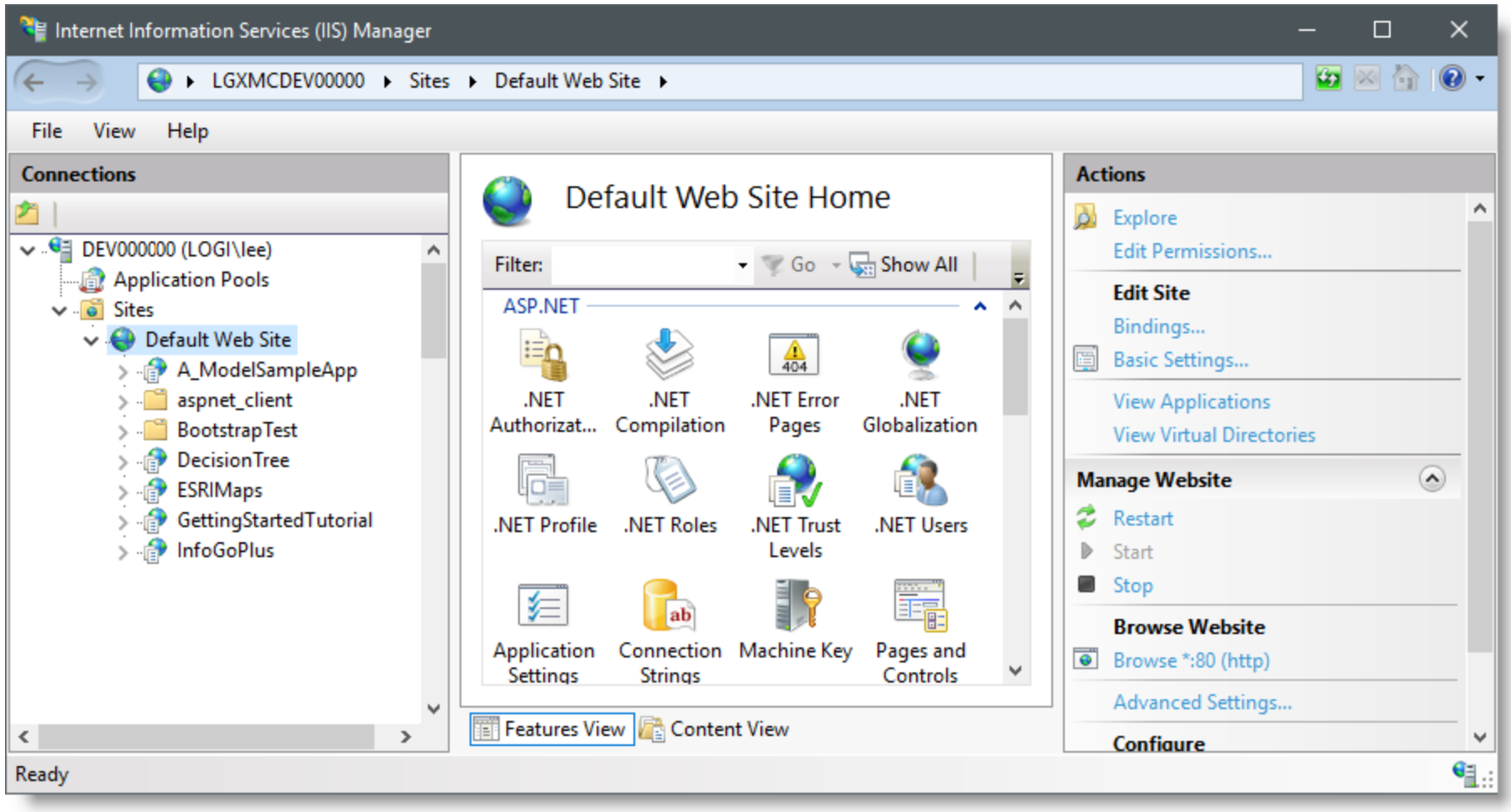




Logi applications are registered in IIS as a *web application*. A web application equates a URL with the physical location of application files and folders on the file system. Logi applications, like other web applications registered in IIS, are accessible from a web browser using their *virtual path*.

The Logi Studio registration wizard only configures Logi applications under the **Default Web Site** section of IIS. See the manual instructions in one of the following sections in order to create an application under other IIS web sites you may have defined, or if you have renamed the Default web site.

You must be logged-in with an account that has administrator privileges at the time Studio attempts the registration; admin privileges are required in order to create and configure an IIS application.



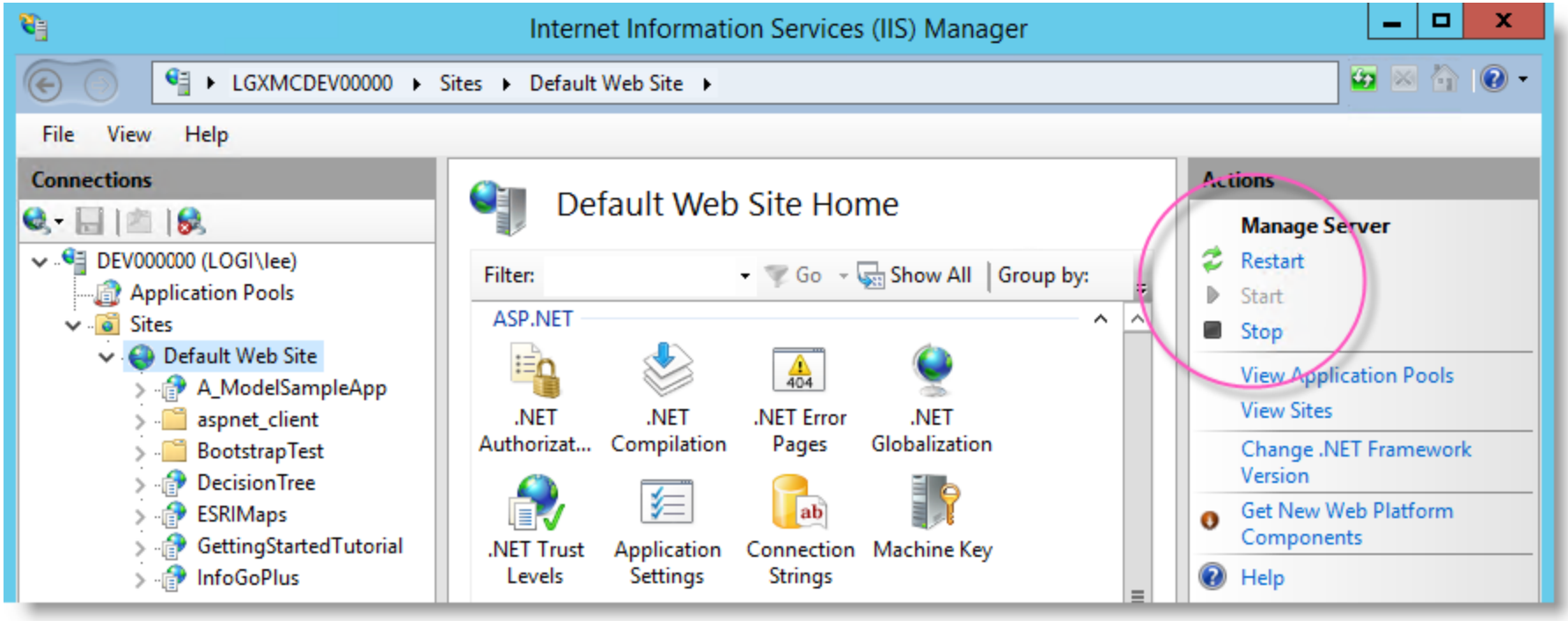
As shown above, in the **IIS 10 Manager** tool, application pools, sites, applications, and folders are listed on the left. Removing an application or virtual directory from the list *does not* remove the physical application folder and files from your drive. Studio's registration wizard accomplishes the following:

- If the appropriate version of the .NET framework is not present, it will be installed, with your consent, and configured for the web server.
- If it's not already running, the Default Web Site will be started.
- A new web application will be created pointing to the Logi application folder.
- The new application will have its .NET version set appropriately.
- The local Windows account "Everyone" will be given Full Control file access permissions to the standard Logi application folders rdDataCache and rdDownload in the application.
- The application pool's Managed Pipeline Mode will be configured as Integrated mode. and Application Pools that use .NET 4.x are required. If such an application pool doesn't exist, Studio will generate a warning when you upgrade an existing Logi application. When you create a new application, Studio will create a new application pool, named "Logi Info .Net v4.x", and will assign the application to it.

The following topics provide instruction for accomplishing some of these tasks manually, in case a situation occurs in which this is necessary, or in case developers which to ensure that the correct configuration did in fact occur.

Starting the Default Web Site

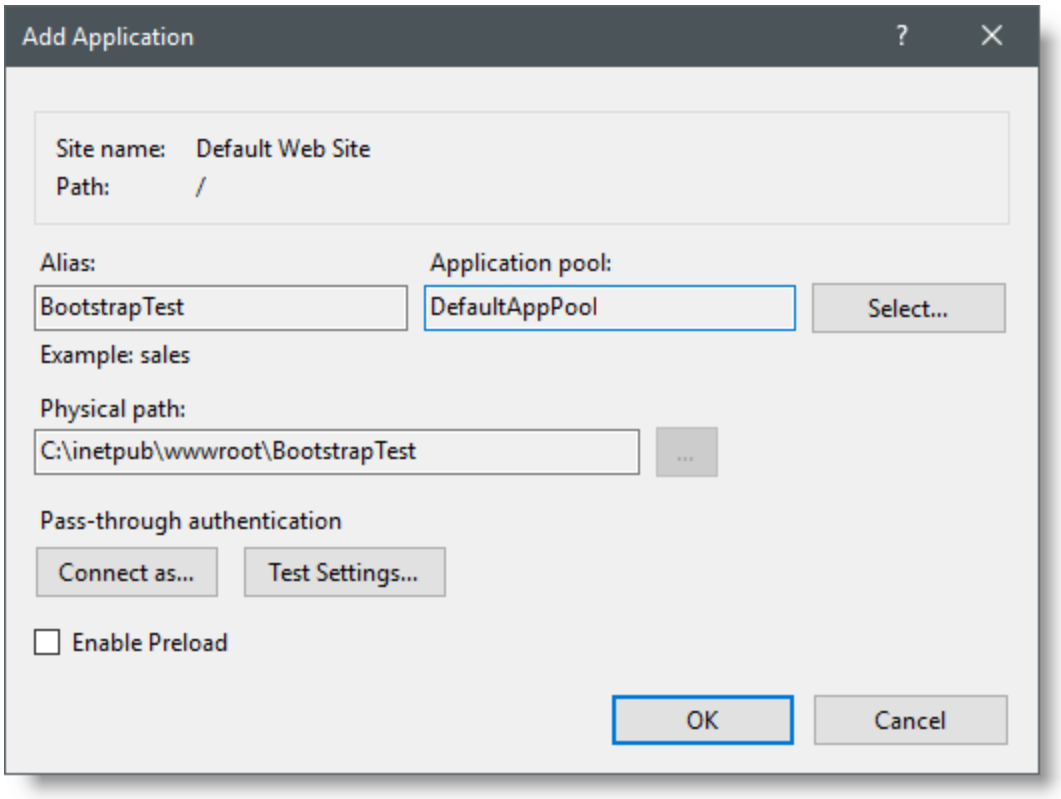
To start the IIS "Default Web Site", open the IIS Manager tool:



As shown in the **IIS 8 Manager** example above, locate and select the Default Web Site. To start the site, if it's stopped, click the **Start** button in the toolbar, which is circled above. Or, you can right-click the web site entry and start or stop it from the pop-up context menu that appears.

Manually Creating an IIS Web Application

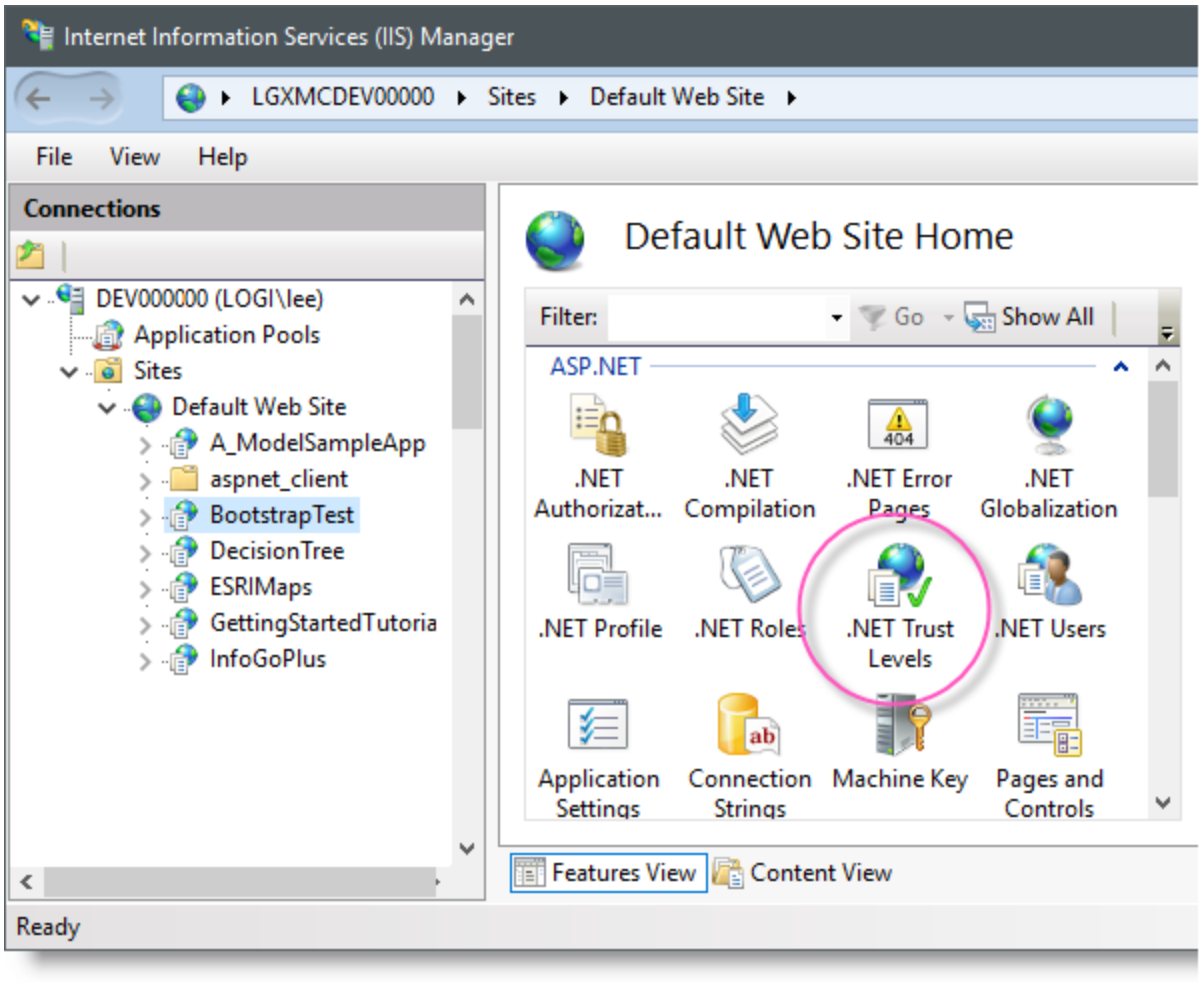
In some circumstances you may need to, or want to, create an IIS web application manually. To do so, open the IIS Manager tool and examine the list of folders and applications beneath the Default Web Site. If you find it, right-click it and select *Convert to Application* from the pop-up context menu.

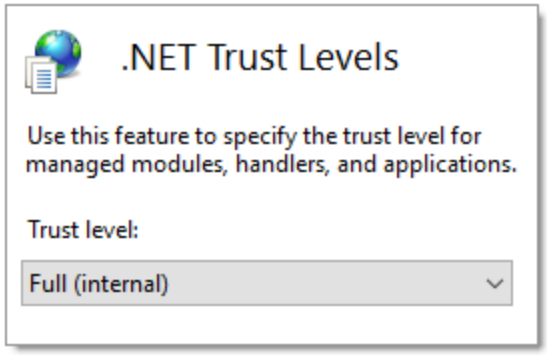


As shown above, fill-in an **Alias** (this will be used in the URL to run this application) and select an **Application Pool** (use the *DefaultAppPool* selection shown unless you've created a special pool for this application). The enter, paste, or browse to the

correct Physical Path for you Logi app's root folder. Click OK. Ensure that the new entry was added to the list of applications beneath the correct web site.

You may want to create a separate Application Pool if you want to isolate this application from others being run on the same web site. That can be useful if you have to stop/restart your application but want to do so without kicking all other users off the web site. More information about Application Pools is available from Microsoft.





The icon for your new web application will have changed from a folder to an app icon. Select your new application in the list and, from the **ASP.NET** options section in the center of the screen, double-click **.NET Trust Levels** icon, shown circled above. Ensure that the Trust Levels setting is **Full (internal)** and, if you had to make a changed, click **Apply** in the right hand Actions panel.

Logi Info Error Messages

This topic describes selected error messages generated by the Logi Engine and web server.

Logi Engine Error Messages

"Access denied for user 'xyz'"
This is typically the result of a failure to connect to a database or server using the supplied credentials. Try using another method outside of your Logi application to manually connect directly, using the same credentials, to verify that they're still valid.

"Chart definition could not be loaded"
This error is commonly seen when the user session has ended or changed, preventing the cache file from being found. There is also a known issue with Windows Server 2012 R2 and Windows updates KB3007507 or KB3000850. These updates can incorrectly trigger within your server causing all the cache files to be removed. Contact Logi Support for additional information.

"Culture 'xx-XX' is not supported"
This error is most often caused by the user's browser not being configured for the indicated language setting. You may be able to resolve it by installing the appropriate language pack on your server, or by using a Globalization element (see *The _Settings Definition*) to override the user's browser culture setting. It's also a known issue for Logi Info versions earlier than 11.4.

"Dashboard/Panel elements must have a unique ID. <panel ID> is duplicated"
This error is displayed when Dashboard Panel elements are not given *unique* element IDs. When working with Dashboards and Extra Gallery Files, this error will also be displayed if the element IDs of Dashboard Panel elements used in the Dashboard and of those panels saved in the gallery file aren't *unique*.

"Element has not been rendered"

This error message can occur when exporting to PDF using SSL and the charts being exported do not have a specific height and width set. Set a specific height and width for the charts.

"Error occurred while parsing EntityName"

This error is caused by an XML parsing error, most often due to a special character, such as an unencoded "&", in the data. Ajax refreshes are particularly vulnerable to this. Solutions include identifying the character and replacing it the datalayer with an encoded version, for example replacing & with & in a SQL query or by using a Calculated Column on the datalayer.

"Index was outside the bounds of the array"

This is a generic error that could be caused by a variety of problems, including the data that's being returned, and configuration or customization changes made within the application. It's often caused by invalid configurations or data with PDF exports. You can try remarking different sections or elements in your definition in order to isolate and identify the offending element, and then check its attributes. Contact Logi Support for further assistance in debugging and resolving the issue.

"Invalid URI: The format of the URI could not be determined"

This error can occur when working with a wizard that retrieves the data columns for a table, chart, etc. When the wizard runs, it attempts to execute the current report definition in order to retrieve the data and its schema. If the report definition lacks critical configuration values, the wizard fails and this error message appears. Typical issues include missing Data Table IDs, errors in SQL queries, andmissing request tokens needed for expressions. To find the offending issue, try Previewing the report to get a more specific error message.

"Name cannot begin with the 'x' character, hexadecimal value 0xnn

Certain reserved characters are expected to be encoded within your data for XHTML to be compiled correctly, including:

Name	Normal	Encoded
apostrophe	'	&apos;
ampersand	&	&amp;
quote	"	&quot;
less than	<	&lt;
greater than	>	&gt;

If the unencoded versions of these characters are found in the data, especially during an Ajax refresh, an error will occur.

"rdProcess request parameter not supplied, or is empty."

A URL was issued to call a Process task, possibly using Action.Process, but it did not include a value for the *rdProcess* query string argument. Typical causes include:

- Failure to properly configure the Action.Process element attributes (no Process Definition File value).
- URL request parameters that exceed the query string limits of the browser or server. For more information, see the Query String Limits section of *Pass Information*.
- If you're using a Startup Process, a bug is causing the named task to fail to execute. For more information, see*The _Settings Definition*.
- Invalid SQL statement used with Event Logging.

"Security is enabled, but the current user cannot be determined"

When using AuthNT security, IIS must be configured to pass the currently logged-in user ID to the Logi Info application. This is

done by adding or enabling Windows Security through the IIS Authentication settings. This error indicates that Anonymous Authentication has not been *disabled* and/or Windows Authentication has not been *enabled* in IIS for the application.

"Selected folder does not appear to be a valid Logi Info application folder"
Logi Studio is unable to open the designated folder as a Logi application because it lacks one or more standard child folders, such as `bin`, `rdTemplate`, or `rdDataCache`, or root files. The folder(s) may not exist or file access permissions may not be set properly on them. One easy solution is to give the Everyone group *Full Control* file access permissions for the Logi application folder (and all of its child folders and files). For more information, see "File Access Permissions" on page 308.

"There was an error calling a plugin. Exception has been thrown by the target of an invocation."
This frequently occurs if the element that calls the plug-in has been configured incorrectly, refer to the documentation for the correct values. Another common cause is that the file access permissions for the folder that contains the plug-in are incorrect - try setting them to grant "Full Control" to the Everyone group. The Debug Trace Page's inner stack trace should provide additional information about the nature of the error, see *Debug Reports*.

"There was an error calling a plugin. There was an error inside a plugin."
If this error occurred when first using an add-on module, such as the SSRM/SSM, or the InfoGo application, you may have a version mismatch. Ensure that the proper versions of Logi Info and add-on modules are installed, see *Add-on Modules*. Another cause can be corrupted bookmark or goCollection files. Otherwise, if you've written your own custom plug-in and receive this message, it's an indication that there's something wrong in your plug-in code. The Debug Trace Page will offer more detailed information about the cause of the error, see *Debug Reports*. Check the .NET plug-in (see *Create .NET Plug-in*) and Java plug-in (see *Create a Java Plug-in*) documentation for guidance in debugging your code. The AddDebugMessage() method available in the rdServerObject is especially useful for inserting your own debug information into the Debug Trace Page during development.

"Wizard cannot continue. Data at the root level is invalid: Line 1, position 1."
This error occurs when using the Theme Editor in Logi Studio, when the _Settings definition has been obfuscated. Obfuscation is

primarily designed for deployed versions of an application, not for development work. The Theme Editor needs to access the _Settings definition to modify a theme. De-obfuscate the definition in order to allow the Theme Editor to work.

"Wizard cannot continue. Filename: xyz. Error: Cannot write configuration file"
This error occurs when Logi Studio is trying to register, or update the version of, a Logi Info application. The most common cause is insufficient OS file privileges, usually the result of not running Studio as an Administrator. You can also try giving the "Everyone" group Full Control security permissions for the application folder.

Web Server Error Messages

"Server Error in '/' Application. The resource cannot be found. HTTP 404."
The web server is unable to find the Logi application. Check the application's _Settings definition's **Path** element and ensure that the Application Path attribute value is correct. When developing on your own machine, this value should be something like *http://localhost/yourAppName*

Accessible Logi Applications

This topic discusses features in Logi managed reporting products that make applications accessible to users with cognitive impairments or disabilities.

This topic contains the following sections:

About Section 508 and Accessibility

The **Section 508** amendment to the Rehabilitation Act of 1973, a law in the United States, was enacted to eliminate barriers in information technology, to make available new opportunities for people with disabilities, and to encourage development of technologies that will help achieve these goals. The law applies to all U.S. Federal agencies when they develop, procure, maintain, or use electronic and information technology.

Section 508 raises awareness of the need to be able to create "accessible" software in order to make it available to users with cognitive impairments or other disabilities. The W3C's Web Accessibility Initiative defines the following requirements. Accessible software needs to be: **Perceivable**

- Provide text alternatives for non-text content.
- Provide captions and other alternatives for multimedia.
- Create content that can be presented in different ways, including by assistive technologies, without losing meaning.
- Make it easier for users to see and hear content.

Operable

- Make all functionality available from a keyboard.
- Give users enough time to read and use content.
- Avoid content that causes seizures.
- Help users navigate and find content.

Understandable

- Make text readable and understandable.
- Make content appear and operate in predictable ways.
- Help users avoid and correct mistakes.

Robust

- Maximize compatibility with current and future user tools.

Logi Info provides a development environment capable of creating 508-compliant applications. However, it's the developer's responsibility when building reports to ensure that they create 508-compliant applications. The HTML output generated by the Logi Engine can be crafted by the developer using Logi Studio to produce accessible reports and analyses.

In broad terms, this means, that developers shouldn't use elements that wouldn't pass 508-compliance testing. These include, for example, Java applets and Flash objects. Reporting and BI rely heavily on "visual" presentations, so building a 508-compliant BI application is restrictive, but very possible.

Logi Info is used in many government agencies to produce applications that are fully-compliant with 508 standards. In addition, several of our customers have commissioned 3rd-party audits of our products, which we've passed, to ensure their 508-compliance.

Specific Accessibility Features

In addition, the following specific accessibility features are available in Logi Info:

Elements	Description
Analysis Grid Charts	Charts in the Analysis Grid now display Chart Title as an alternate text in HTML <desc> and <title> tags.
Data Table	Includes an **Accessible Headers** attribute which, when enabled, adds the `headers` attribute to the `<TD>` tag. This is another way to associate data cells and headers, even though `scope` is usually sufficient for most tables. In very complex tables, `scope` may cause table headers to apply to (be "scoped for") cells that aren't associated with that header. In these cases, `headers` and `id` may make the table accessible to a screen reader user. With this approach, every `<TH>` tag is assigned a unique `id` attribute value. Then, each and every `<TD>` cell within the table is given a `headers` attribute with values that match each `<TH> id` value the cell is associated to.
Data Table Crosstab Table	Includes an **Accessible Summary** attribute, which adds the `summary` attribute to the `<TABLE>` tag. The arbitrary text you enter in this element attribute, which usually describes the table's contents or purpose and functions like an extended table title, is consumed by screen readers. It has no visual effect in browsers. For tables created in the Analysis Grid super-element, the `summary` attribute is set to *AnalysisGrid DataTable* and *AnalysisGrid CrosstabTable*, respectively, by default.
Data Table Column	Includes a **Scope Row Header** attribute which, when enabled, includes header information for each HTML row generated, changing `<TD>` tags to `<TH scope="Row">` tags. This attribute tells the browser and screen reader that everything within a column that is associated to the header with `scope="col"` in that column, and

Elements	Description
	that a cell with `scope="row"` is a header for all cells in that row.
Fieldset Box	A container element that wraps its contents in a single-line rectangle with a caption at the top left. Facilitates use of screen readers, in particular with grouped input elements such as Input check box and Input Radio Buttons.
Image	Unless an **Alternate Text** attribute value is provided, this element generates a tag that includes a blank `alt` attribute: `<IMG alt="">`
Interactive Paging	Includes four attributes to support Alternate Text: The **First Page Alternate Text**, **Last Page Alternate Text**, **Next Page Alternate Text**, and **Previous Page Alternate Text** attributes specify text to be displayed by browsers with their image display disabled and by screen readers. Their default values are, respectively, *First page*, *Last page*, *Next page*, and *Previous page*.

Label	Includes two attributes for improved accessibility: Label caption text is usually generated in our HTML as plain text. With the use of the **Html Tag** attribute, the text will be generated enclosed in opening and closing tags of the specified variety. You can select a tag type from a list of common options or just type one in (just the tag name, do not include its < and > brackets). The **For** attribute improves accessibility with some browsers. When a Label is used as the caption of an input element, the For attribute specifies which form element the label is bound to. Set it to the ID of its related input element. Here's an example of the generated HTML when this attribute is given a value of *male*:
	
	`<label for="male">Male</label>`
	`<input type="radio" name="sex" id="male" value="male">`

Elements	Description
	A benefit of using Labels for input captions is that the user can click on the label itself to set focus to the form element. This is useful to some with motor disabilities, particularly when selecting small check boxes and radio buttons.
PDF Output	The Action.PDF export process allows you to add PDF tags or set properties of different attributes. Document options on PDF export element to set include: images, labels, and paragraphs of the exported document.
Report Root	Includes a **Language** attribute which can be used to declare the language of a web page, in order to assist search engines and browsers. For example, once this attribute is set, this HTML will be generated: `<HTML lang="fr" xml:lang="fr">` The language should be specified using one of the ISO 639-1 language codes.

User Input	If a **Caption** attribute value is provided, all User Input elements generate a **Label** tag with a **For** attribute to display the caption. This associates the caption with the UI element. For example, the HTML generated might be:
	
	`<label for="name">Name:</label>`
	`<input id="name" type="text" name="textfield">`
	
	A benefit of using Labels for input captions is that the user can click on the label itself to set focus to the form element. This is useful to some with motor disabilities, particularly when selecting small check boxes and radio buttons.

Glossary

A

API

API, short for Application Program Interface, is a set of routines, protocols, and tools for building software applications. In business intelligence, APIs may be used to enable end-users to directly update source systems.

Authentication

Authentication is the verification of a user's identity.

Authorization

After a user's identity has been authenticated, authorization grants or denies access to reports, columns, and records to selected users or user-groups.

B

Big Data

Refers to both the ever-growing volumes of data in use today and also to services that are specifically engineered to provide and manipulate very large data volumes.

Business Analytics

Business analytics, or business intelligence (BI), gives customers the ability to rapidly create scalable, interactive data analysis applications, and self-service capabilities users can access from anywhere and on any device.

C

Columnar Data Store

Columnar data store is a type of big data repository containing structured data in columns and rows. The main benefits are that the data can be highly compressed and is easily searchable.

CRM

A Customer Relationship Management (CRM) system is a database-based system that records a company’s daily customer-related transactions. CRMs can help customer representatives to provide better service, close more deals, and increase revenue.

CSS

Cascading Style Sheets (CSS) is a technology that allows the presentation aspects of web pages to be separated from the page content. It can be used to add "styling" (e.g. apply fonts, colors, alignment, spacing, and more) to web pages.

D

Data Discovery

Data discovery is the capability to analyze data on-the-fly and uncover insights from it.

Data Enrichment

Data enrichment is a method of preparing data to make it ready for analysis and exploitation, and can include formatting, adding calculations, joining with other data, and more.

DevNet

The Logi Developer Network website.

Drill Down

Drill Down is a capability that allows the user to get a view of the underlying or supporting data used in an analysis.

Drill Through

Drill Through is similar to Drill Down but takes it one step further by applying analysis to the underlying or supporting data.

E

Elemental Development

A development approach used in Logi Info that lets developers build feature-rich applications by using reusable, pre-built elements, rather than by writing low-level code.

F

Forecasting

A technique involving data mining and analysis leading to predictions about what will happen in the future.

G

Geo Mapping

The combination of geographic and other data to produce map visualizations, such as Google or Leaflet maps.

H

Heatmap

A Heatmap chart, sometimes called a "tree map", which uses a unique arrangement of rectangles to represent data and relationships, using color and size.

I

Interpolation

The process of evaluating a literal value match containing one or more placeholders, yielding a result in which the placeholders are replaced with their corresponding values.

J

JavaScript

JavaScript is a programming language supported by the majority of modern web browsers and used by many websites.

JDBC

Java Database Connectivity (JDBC) is an API used to access relational databases. Open Database Connectivity (ODBC) is a similar API designed for use with Java.

JSON

JavaScript Object Notation (JSON) is a lightweight data-interchange format that's easy for humans to read and write, and easy for computers to parse and generate.

K

KPI

Key Performance Indicators (KPIs) are visual indicators, in the form of color-coded shapes, which are tied to a pre-defined, critical threshold.

L

LDAP

The Lightweight Directory Access Protocol (LDAP) is an Internet protocol applications use to look up information from a server and is frequently used for containing user login information.

M

My Term

My definition

N

NoSQL

"Not only SQL" (NoSQL) is an alternative to traditional relational databases, and doesn't rely on tables and a pre-determined schema. NoSQL databases are especially useful for working with large sets of distributed data.

O

ODBC

Open Database Connectivity (ODBC) is an API used to access relational databases. Java Database Connectivity (JDBC) is a similar API designed for use with Java.

OLAP

Online Analytical Processing (OLAP) is the process of analyzing data stored in multi-dimensional "cubes".

R

REST

Representational State Transfer (REST) is a type of API used to provide interoperability between computer systems on the Internet.

S

SSM

The Self-Service Module (SSM) is a package that includes Logi Info + SSRM + Discovery or Logi Platform Services.

SSRM

The Self-Service Reporting Module (SSRM) is a Logi Info add-on module that adds special elements to Info and includes the InfoGo application.

W

Write-Back

The ability to update data sources, typically by adding, editing, or deleting data.