



Reference for Operational and Actionable Resources for Security

Version 2023.3
2024-05-16

Reference for Operational and Actionable Resources for Security
InterSystems IRIS Data Platform Version 2023.3 2024-05-16
Copyright © 2024 InterSystems Corporation
All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble®, InterSystems HealthShare®, InterSystems IRIS®, and TrakCare are registered trademarks of InterSystems Corporation. HealthShare® CMS Solution Pack™ HealthShare® Health Connect Cloud™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, and InterSystems TotalView™ For Asset Management are trademarks of InterSystems Corporation. TrakCare is a registered trademark in Australia and the European Union.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)

Tel: +1-617-621-0700

Tel: +44 (0) 844 854 2917

Email: support@InterSystems.com

Table of Contents

Securing Your Instance	1
Introduction to Securing Your Instance	2
Prepare for InterSystems Security	3
System Management and Security	13
Tighten Security for an Instance	25
Security Advisor	36
Secure InterSystems Processes and Operating-System Resources	39
Security Checklist	45
Identity and Access Management	51
Identity and Access Management Introduction	52
Kerberos Authentication	53
Operating System–Based Authentication	64
Instance Authentication	65
Delegated Authentication	68
Two-Factor Authentication	78
JSON Web Token (JWT) Authentication	87
LDAP	90
OAuth 2.0 and OpenID Connect	113
Delegated Authorization	166
Advanced Topics in Authentication	173
Encryption	177
Introduction to InterSystems Encryption	178
Key Management Tasks	179
Using Encrypted Databases	200
Data-Element Encryption	210
Protecting Against Data Loss	213
Handling Emergency Situations	214
Additional Encryption Information	224
FIPS-2 Compliance	226
Cryptographic Standards and RFCs	228
Public Key Infrastructure	229
Demo: Database Encryption	239
TLS	247
TLS with the Superserver	248
TLS with Telnet	249
TLS with Python Clients	250
TLS with Java Clients	252
TLS with .NET Clients	257
TLS with Studio	258
TLS and Windows with .ini File	259
Configuring InterSystems IRIS to Use TLS with Mirroring	265
TLS with TCP Devices	268
TLS with the Web Gateway	272
Certificate Chain	273

List of Figures

Figure B–1: Architecture of a Kerberos-Protected Web Connection 56

Figure B–2: A TOTP Issuer, Account, Key, and QR Code 80

List of Tables

Table A-1: Enabled Services 8

Table A-2: Required Public Resources and Their Permissions 32

Table B-1: Connection Tools, Their Access Modes, and Their Services 55

Table D-1: Valid Certificate Distribution Schemes 273

Securing Your Instance

Introduction to Securing Your Instance

Introduction to Securing Your Instance

Because security requires actions within an instance and in an instance's larger environment, InterSystems IRIS provides both guidance and tools to help [secure an instance](#). These include:

- A checklist of topics to review as you prepare to deploy InterSystems IRIS.
- A guide and tools for securing an instance using its built-in features.
- A checklist for hardening security for an instance at the operating-system level and by managing its processes.

Prepare for InterSystems Security

Prepare for InterSystems Security

The material in this section describes some of the security-related issues you need to consider before installing InterSystems IRIS. For an overview of the InterSystems security features, see “About InterSystems Security”; you may also want to review details about [authentication](#) or [authorization](#).

This section covers the following topics:

- [Initial InterSystems Security Settings](#) — Describes the characteristics of the different default security settings. It is particularly useful if you choose to use Normal or Locked Down InterSystems security.
- [Configure User Accounts](#) — Discusses the necessary permissions for a user account that runs InterSystems IRIS.
- [Prepare the Security Environment for Kerberos](#) — Details the additional tasks you need to perform if you are planning on using Kerberos as an authentication mechanism with InterSystems IRIS. If you are not using Kerberos in your environment, you can bypass this topic.

Important: If your security environment is more complex than those this document describes, contact the [InterSystems Worldwide Response Center \(WRC\)](#) for guidance in setting up such an environment.

After reading *About InterSystems Security* and following the procedures in this section, you are prepared to provide the pertinent security information to the installation procedure, as described in the Installation Guide.

Initial InterSystems Security Settings

During installation, there are three initial security configurations to choose from: **Minimal**, **Normal**, or **Locked Down**. A good rule of thumb is to choose Locked Down for instances to be used in production environments and Normal for instances to be used in development environments. The following sections describe the differences between these configurations, as well as the initial service properties for each configuration:

- [Initial User Security Settings](#)
- [Initial User Account Passwords](#)
- [Initial Service Properties](#)

For production environments, you should adjust the individual security settings after installation, regardless of which option you choose. For more information see the following sections:

- [Tighten Security for an Instance](#)
- [Security Advisor](#)
- [Secure InterSystems Processes and Operating-System Resources](#)
- [Checklist for Hardening Your Deployment](#)

Important: If you are concerned about the visibility of data in memory images (often known as core dumps), see [Protect Sensitive Data in Memory Images](#).

Initial User Security Settings

For general information about InterSystems IRIS user accounts, see [User Accounts](#).

All user accounts share certain password requirements and settings. The initial values for these settings are based on which security level you choose, as described in the following table:

Security Setting	Minimal	Normal	Locked Down	Description
Password Pattern*	3.32ANP	3.32ANP	8.32ANP	By default, passwords allow alphanumeric characters and punctuation. The initial length requirement is 3 to 32 characters for Minimal and Normal installations, or 8 to 32 for Locked Down installations. For more information about password patterns, see Password Strength and Password Policies .
Inactive Limit*	0	90 days	90 days	The Inactive Limit is the number of days an account can be inactive before it is disabled. For Minimal installations, the limit is set to 0 indicating that accounts are never disabled, no matter how long they are inactive. Normal and Locked Down installations have the default limit of 90 days.
Enable _SYSTEM User	Yes	Yes	No	
Roles assigned to UnknownUser	%All	None	None	When an unauthenticated user connects, InterSystems IRIS assigns a special name, <code>UnknownUser</code> , to <code>\$USER-NAME</code> and assigns the roles defined for that user to <code>\$ROLES</code> . In a Minimal security installation, the <code>UnknownUser</code> is assigned the <code>%All</code> role; <code>UnknownUser</code> has no roles when choosing a security level other than Minimal. For more details on the use of <code>\$USERNAME</code> and <code>\$ROLES</code> , see Users and Roles .

* You can maintain these settings from the **System > Security Management > System Security Settings > System-wide Security Parameters** page of the Management Portal. See [System-wide Security Parameters](#) for more information.

Initial User Account Passwords

InterSystems IRIS creates multiple user accounts during installation. The [predefined InterSystems IRIS user accounts](#) have different default passwords and behavior depending on whether an installation uses Minimal security, Normal security, or Locked Down security. These differences are as follows:

- *Minimal security* – All the created accounts except _PUBLIC have an initial default password of “SYS”. With the exception of UnknownUser, you should change the account passwords after installation in order to prevent unauthorized access to your InterSystems IRIS instance.

The _PUBLIC account has no password by default and should never be given a password, since it is never enabled.

- *Normal security* – All the created accounts except _PUBLIC receive the same password as is chosen for the privileged user account. It is recommended that you change these passwords after installation, so that each account has its own password.

The _PUBLIC account has no password by default and should never be given a password, since it is never enabled.

- *Locked Down security* – All the created accounts except _PUBLIC receive the same password as is chosen for the privileged user account. It is recommended that you change these passwords after installation, so that each account has its own password.

The _PUBLIC account has no password by default and should never be given a password, since it is never enabled. In Locked-Down installations, the _SYSTEM account is also disabled.

CAUTION: The default password is a security vulnerability, particularly in a Minimal Security installation. To address this issue, disable the accounts or change their passwords. InterSystems recommends disabling the account.

This is a critical concern with containerized instances in particular; see [Authentication and passwords](#) for more information, including ways in which you can address the issue.

Initial Service Properties

Services are the primary means by which users and computers connect to InterSystems IRIS. For detailed information about the InterSystems services see [Services](#).

Service Property	Minimal	Normal	Locked Down	Description
------------------	---------	--------	-------------	-------------

Service Property	Minimal	Normal	Locked Down	Description
Use Permission is Public	Yes	Yes	No	If the Use permission on a service resource is Public, any user can employ the service; otherwise, only privileged users can employ the service.
Requires Authentication	No	Yes	Yes	For installations with initial settings of Normal or Locked Down, all services require authentication of some kind (Instance Authentication, operating-system-based, or Kerberos). Otherwise, unauthenticated connections are permitted.

Service Property	Minimal	Normal	Locked Down	Description
Enabled Services	Most	Some	Fewest	The initial security settings of an installation determine which of certain services are enabled or disabled when InterSystems IRIS first starts. The Enabled Services table below shows these initial settings.

Table A-1: Enabled Services

Service	Minimal	Normal	Locked Down
%Service_Bindings	Enabled	Enabled	Disabled
%Service_CacheDirect	Enabled	Disabled	Disabled
%Service_CallIn	Enabled	Disabled	Disabled
%Service_ComPort	Disabled	Disabled	Disabled
%Service_Console*	Enabled	Enabled	Enabled
%Service_ECP	Disabled	Disabled	Disabled
%Service_Monitor	Disabled	Disabled	Disabled
%Service_Telnet*	Disabled	Disabled	Disabled
%Service_Terminal†	Enabled	Enabled	Enabled
%Service_WebGateway	Enabled	Enabled	Enabled

* Service exists on Windows servers only

† Service exists on non-Windows servers only

Configure User Accounts

During the installation process, you must choose an account to run the InterSystems IRIS process as the instance owner. The installation creates an InterSystems IRIS account with the %All role for the instance owner, providing that account with full administrator access to InterSystems IRIS.

To ensure that the instance owner has the necessary privileges, you may need to create a new user account. The following sections contain OS-specific details about what accounts and privileges are necessary:

- Windows — [Windows User Accounts](#) in the “Installing InterSystems IRIS on Microsoft Windows” chapter of the Installation Guide.
- Unix® and Linux — [Determine Owners and Groups](#) in the “Installing InterSystems IRIS on UNIX®, Linux, and macOS” chapter of the Installation Guide.

Prepare the Security Environment for Kerberos

All InterSystems IRIS supported platforms have versions of Kerberos supplied and supported by the vendors. To use Kerberos, you must have either a Kerberos key distribution center (KDC) or a Windows domain controller available on your network. The installation preparations for each are as follows:

- Windows domain controller

This configuration uses a Windows domain controller for KDC functionality with InterSystems IRIS servers and clients on Windows and non-Windows machines. A domain administrator creates domain accounts for running the InterSystems services on InterSystems IRIS servers. See the following sections for the requirements for using both Windows and non-Windows InterSystems IRIS servers:

- [Create Windows Service Accounts for Windows Servers](#)
- Depending on the applications in use on your system, you may also need to perform actions described in [Configure Windows Kerberos Clients](#).
- [Create Windows Service Accounts for Non-Windows Servers](#)

- Non-Windows KDC

This configuration uses a UNIX® or Kerberos KDC with InterSystems IRIS servers and all clients on non-Windows machines. See the following two sections for the requirements for using a UNIX® or macOS KDC and InterSystems IRIS servers:

- [Create Service Principals on a KDC for Non-Windows Servers](#)
- [Test Kerberos KDC Functions](#)

A Note on Terminology

This document refers to related, but distinct entities:

- Service account — An entity within an operating system, such as Windows, that represents a software application or service.
- Service principal — A Kerberos entity that represents a software application or service.

Create Windows Service Accounts for Windows Servers

Microsoft Windows implements the Kerberos authentication protocol by integrating the KDC with other security services running on the domain controller. Before you install InterSystems IRIS in a Windows domain, you must use the Windows domain controller to create a service account for each InterSystems IRIS server instance on a Windows machine.

Account Characteristics

When you create this account on the Windows domain controller, configure it as follows:

- Set the account's **Password never expires** property.
- Make the account a member of the **Administrators** group on the InterSystems IRIS server machine.

- Add the account to the **Log on as a service** policy.

Important: If a domain-wide policy is in effect, you must add this service account to the policy for InterSystems IRIS to function properly.

Names and Naming Conventions

In an environment where clients and servers are exclusively on Windows, there are two choices for naming service principals. You can follow the standard Kerberos naming conventions, which ensures compatibility with any non-Windows systems in the future, or you can use any unique string. Each of these choices involves a slightly different process of configuring a connection to a server.

- For a name that follows Kerberos conventions, the procedure is:
 1. Run the Windows **setspn** command, specifying the name of service principal in the form *service_principal/fully_qualified_domain_name*, where *service_principal* is typically *iris* and *fully_qualified_domain_name* is the machine name along with its domain. For example, a service principal name might be *iris/irissrvr.example.com*. For detailed information on the **setspn** tool, see the [Setspn](#) page in the Microsoft documentation.
 2. In the **InterSystems IRIS Server Manager** dialog for adding a new preferred server, choose Kerberos. What you specify for the **Service Principal Name** field should match the principal name specified in **setspn**.
- For a name that uses any unique string, the procedure is:
 1. Choose a name for the service principal. A suggested naming convention for each account representing an InterSystems IRIS server instance is “*irisHOST*”, which is the literal *iris* followed by the host computer name in uppercase. For example, if you are running an InterSystems IRIS server on a Windows machine called WINSRV, name the domain account *irisWINSRV*.
 2. In the **InterSystems IRIS Server Manager** dialog for adding a new preferred server, choose Kerberos. Specify the selected name for the service principal in the **Service Principal Name** field.

For more information on configuring remote server connections, see [Connecting to Remote Servers](#) for the detailed procedure.

Configure Windows Kerberos Clients

If you are using Windows clients with Kerberos, you may also need to configure these so that they do not prompt the user to enter credentials. This is required if you are using a program that cannot prompt for credentials — otherwise, the program is unable to connect.

To configure Windows not to prompt for credentials, the procedure is:

1. On the Windows client machine, start the registry editor, **regedit.exe**.
2. Go to the HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Kerberos\Parameters key.
3. In that key, set the value of *AllowTgtSessionKey* to 1.

Create Windows Service Accounts for Non-Windows Servers

Before you install InterSystems IRIS in a Windows domain, you must use the Windows domain controller to create a service account for each InterSystems IRIS server instance on a non-Windows machine. Create one service account for each machine, regardless of the number of InterSystems IRIS server instances on that machine.

A suggested naming convention for these accounts is “*irisHOST*,” which is the literal, *iris*, followed by the host computer name in uppercase. For example, if you run an InterSystems IRIS server on a non-Windows machine called UNIXSRV, name the domain account *irisUNIXSRV*. For InterSystems IRIS servers on non-Windows platforms, this is the account that maps to the Kerberos service principal.

Important: When you create this account on the Windows domain controller, InterSystems IRIS requires that you set the **Password never expires** property for the account.

To set up a non-Windows InterSystems IRIS server in the Windows domain, it must have a keytab file from the Windows domain. A keytab file is a file containing the service name for the InterSystems IRIS server and its key.

To accomplish this, map the Windows service account (`irisUNIXSRVR`, in this example) to a service principal on the InterSystems IRIS server and extract the key from the account using the **ktpass** command-line tool on the domain controller; this is available as part of the Windows support tools from Microsoft.

The command maps the account just set up to an account on the UNIX®/Linux machine; it also generates a key for the account. The command must specify the following parameters:

Parameter	Description
<code>/princ</code>	The principal name (in the form <i>iris/<fully qualified hostname>@<kerberos realm></i>).
<code>/mapuser</code>	The name of the account created (in the form <i>iris<HOST></i>).
<code>/pass</code>	The password specified during account creation.
<code>/crypto</code>	The encryption type to use (use the default unless specified otherwise).
<code>/out</code>	The keytab file you generate to transfer to the InterSystems IRIS server machine and replace or merge with your existing keytab file.

Important: The principal name on UNIX®/Linux platforms must take the form shown in the table with the literal `iris` as the first part.

Once you have generated a key file, move it to a file on the InterSystems IRIS server with the [key file characteristics](#) described in the section below.

Create Service Principals on a KDC for Non-Windows Servers

In a non-Windows environment, you must create a service principal for each UNIX®/Linux or macOS InterSystems IRIS server that uses a UNIX®/Linux or macOS KDC. The service principal name is of the form *iris/<fully qualified hostname>@<kerberos realm>*.

Key File Characteristics

Once you have created this principal, extract its key to a key file on the InterSystems IRIS server with the following characteristics:

- On most versions of UNIX®, the pathname is *install-dir/mgr/iris.keytab*. On macOS and SUSE Linux, the pathname is */etc/krb5.keytab*.
- It is owned by the user that owns the InterSystems IRIS installation and the group `irisusr`.
- Its permissions are `640`.

Test Kerberos KDC Functions

When using Kerberos in a system of only non-Windows servers and clients, it is simplest to use a native UNIX®/Linux KDC rather than a Windows domain controller. Consult the vendor documentation on how to install and configure the KDC; these are usually tasks for your system administrator or system manager.

When installing Kerberos, there are two sets of software to install:

- The KDC, which goes on the Kerberos server machine.

- There also may be client software, which goes on all machines hosting Kerberos clients. This set of software can vary widely by operating system. Consult your operating system vendor documentation for what client software exists and how to install it.

After installing the required Kerberos software, you can perform a simple test using the **kadmin**, **kinit**, and **klist** commands to add a user *principal* to the Kerberos database, obtain a TGT (ticket-granting ticket) for this user, and list the TGT.

Once you successfully complete a test to validate that Kerberos is able to provide tickets for registered principals, you are ready to install InterSystems IRIS.

System Management and Security

System Management and Security

This page covers access to the InterSystems IRIS® data platform Management Portal, and other security-related features of the Portal.

Manage InterSystems IRIS Security Domains

InterSystems security domains provide a grouping of users that corresponds to Kerberos realms and Windows domains. If your instance is using Kerberos, its InterSystems IRIS domain corresponds to a Kerberos realm. If you are using a Windows domain, this also corresponds to a Kerberos realm.

While a security domain name often takes the form of an Internet domain name, there is no requirement that it do so. A security domain name can contain any character except the at sign (@).

Single and Multiple Domains

InterSystems IRIS supports the use of either a single-domain or multiple-domains.

To specify support for a single domain or multiple domains, use the **Allow multiple security domains** field of the **System-wide Security Parameters** page of the Management Portal (**System Administration > Security > System Security > System-wide Security Parameters**), described in the [System-wide Security Parameters](#) section.

For an instance with a single domain:

- The `$USERNAME` variable does not include the domain name.
- System utilities do not show the domain name when displaying usernames.
- It is prohibited to specify a username from any domain other than the default domain (described in the following section).

For an instance with multiple domains:

- The `$USERNAME` variable includes the domain name.
- System utilities show the domain name when displaying usernames. This includes the **Users** page (**Security Administration > Security > Users**).
- Users log in with their fully qualified name on their domain, such as `documentation@intersystems.com`. If there are two accounts that share the initial portion of the fully qualified name and where the domain names differ, then these are stored as two separate user accounts (where each has its own attributes and these attributes can have differing values).
- You cannot edit usernames.

The Default Security Domain

Each instance has a default security domain. This is the domain assumed for any username where no domain is specified. For example, if the default domain is “intersystems.com”, the user identifiers “info” and “info@intersystems.com” are equivalent. When InterSystems IRIS is installed, it uses the local domain name to provide an initial value for the parameter.

For instances with multiple security domains, you can select a new default security domain using the **Default Security Domain** field of the **System-wide Security Parameters** page (**System Administration > Security > System Security > System-wide Security Parameters**), described in the [System-wide Security Parameters](#) section.

List, Create, Edit, and Delete Security Domains

The **LDAP Configurations** page [lists an instance's existing security domains](#) and configurations, allows you to [create domains](#) and configurations, and allows you to [modify](#) or [delete](#) existing ones.

List Security Domains

To see the list of an instance's domains, go to the **Security LDAP Configurations** page (**System Administration > Security > System Security > LDAP Configurations**). For each domain, the page displays:

- **Login Domain Name** — The domain's name. Click this to [edit the domain's properties](#).
- **LDAP Enabled** — Whether or not LDAP connections are enabled for this domain.
- **Description** — The domain's description.
- A **Delete** link — After confirmation, removes a domain from the instance.

Note: If Kerberos is enabled for an instance, the menu choice that gets to this page is **LDAP/Kerberos Configurations**. The name of the page is **Security LDAP/Kerberos Configs**.

Create a Security Domain

To create a domain for the instance to use, create an LDAP configuration that specifies that domain; creating the LDAP configuration creates the domain:

1. Go to the **Security LDAP Configurations** page (**System Administration > Security > System Security > LDAP Configurations**).
2. Click the **Create New LDAP configuration** button. Selecting this displays the **Edit LDAP configuration** page.
3. On the **Edit LDAP configuration** page, enter the **Login Domain Name** and an optional description.
4. Then enter values for other [configuration fields](#) and click **Save** to create the configuration and the domain.

Note: If Kerberos is enabled for an instance, the menu choice that gets to this page is **LDAP/Kerberos Configurations**. The name of the page is **Security LDAP/Kerberos Configs**.

Edit a Security Domain

To edit a domain:

1. Go to the **Security LDAP Configurations** page (**System Administration > Security > System Security > LDAP Configurations**).
2. Click **Login Domain Name** to edit a domain's and its [configuration fields](#).
3. Click **Save** to save the modified configuration and domain.

Note:

1. You cannot modify a domain's name. You can alternately create a new domain with the preferred name and then delete an existing domain.
2. If Kerberos is enabled for an instance, the menu choice that gets to this page is **LDAP/Kerberos Configurations**. The name of the page is **Security LDAP/Kerberos Configs**.

Delete a Security Domain

To delete a domain:

1. Go to the **Security LDAP Configurations** page (**System Administration > Security > System Security > LDAP Configurations**). This displays a list of domains.
2. Click **Delete** in the domain's row.
3. Confirm the deletion.

Note: If Kerberos is enabled for an instance, the menu choice that gets to this page is **LDAP/Kerberos Configurations**. The name of the page is **Security LDAP/Kerberos Configs**.

Password Strength and Password Policies

InterSystems IRIS allows you to specify requirements for user passwords by supplying a string of the form:

`X.Y[ANP]`

where

- *X* is the minimum number of characters in the password.
- *Y* is the maximum number of characters in the password.
- *A*, *N*, and *P* specify whether Alphabetic characters, Numeric characters, and Punctuation characters are permitted in the password.

These rules are based on the ObjectScript pattern matching functionality. This functionality is described in [Pattern Matching](#).

Note: The value for this parameter does not affect existing passwords.

Suggested Administrator Password Strength

Ideally, administrator passwords should be a random mixture of uppercase and lowercase alphabetic characters, numerals, and punctuation. InterSystems strongly recommends a minimum password length of 12 such random characters.

Emergency Access

InterSystems IRIS provides a special emergency access mode that can be used under certain dire circumstances, such as if there is severe damage to security configuration information or if no users with the `%Admin_Manage:Use` or `%Admin_Security:Use` privileges are available (that is, if all users are locked out). Although InterSystems IRIS attempts to prevent this situation by ensuring that there is always at least one user with the `%All` role, that user may not be available or may have forgotten the password.

To obtain emergency access to an instance of InterSystems IRIS, you must either have root or administrator privileges where the instance is running (if it was installed by root) or be the user who installed the instance (if it was not installed by root). This requirement limits emergency access to users who already have sufficient privileges to perform administrative operations on the instance, such as installing a new instance over the existing one.

Emergency access topics:

- [How Emergency Access Works](#)
- [Invoke Emergency Access Mode on Windows](#)
- [Invoke Emergency Access Mode on UNIX®, Linux, and macOS](#)

How Emergency Access Mode Works

When InterSystems IRIS is running in emergency access mode, only a single user (called the *emergency user*) is permitted. This username does not have to be previously defined within InterSystems IRIS. If the instance already has a user account with the same username as the emergency user, then the emergency user has the privileges associated with emergency access mode instead of the privileges for the existing standard user account.

The emergency user account and password are only valid for the single invocation of emergency mode. If the username specified for the emergency user is a previously defined username within your InterSystems IRIS instance, restarting the system into its normal mode restores the original password and security privileges for that user account. If the username

specified for the emergency user is new, InterSystems IRIS saves the login credentials and security privileges for this new user when it restarts into its normal mode, though the user account is disabled.

Tip: To prevent the accumulation of dormant accounts with the **%ALL** role registered in your InterSystems IRIS instance, we recommend using previously defined usernames for the emergency user instead of new usernames. This also allows systems with multiple administrators to track the authorship of changes made while in emergency access mode [via the logs](#), if each administrator initializes emergency access mode using their own username.

In emergency access mode, InterSystems IRIS has the following constraints and behaviors:

- The emergency user is the only permitted user. Any attempt by another user to log in will fail. The emergency user has the **%ALL** role.
- There is only access using Instance Authentication — no other authentication mechanism is supported. Two-factor authentication is disabled. This avoids any situation where two-factor authentication might prevent the emergency user from being able to authenticate.
- For the web applications that control the Portal (`/csp/sys` and `/csp/sys/*`), the standard login page (`%CSPLogin.cls`) is used during emergency access even if there is a custom login page available; this ensures that the emergency user has access to the Portal, since a custom login page may prevent authentication from occurring. For other web applications, if there is a custom login page, then that page is used during emergency login.
- After emergency access login, InterSystems IRIS attempts to audit all events for the active process; InterSystems IRIS start-up proceeds even if this is not possible. Login failures in emergency access mode are not audited.
- Console, Terminal, and Web Gateway (**%Service_Console**, **%Service_Terminal**, and **%Service_WebGateway**) are the only services that are enabled. All other services are disabled. This does not affect the enabled or disabled status of services when InterSystems IRIS starts in non-emergency mode; only the current (emergency), in-memory information about services is affected.
- For the enabled services, only authenticated access is permitted. InterSystems IRIS uses its own password authentication for the services, where the emergency access username and password must be used.
- The emergency user can make changes to the InterSystems IRIS configuration, but these changes are not activated until the next time that InterSystems IRIS is started in normal (not emergency) mode. This is in contrast to the normal operation of InterSystems IRIS, in which configuration changes are primarily activated without restarting InterSystems IRIS.

Invoke Emergency Access Mode on Windows

To start InterSystems IRIS in emergency access mode, the user must be a member of the Administrators group. Then, do the following:

1. Start a command prompt, running it as an administrator. This can either be:
 - The Windows Command Prompt program. Right-click the **Command Prompt** choice in the menu and then choose **Run as Administrator**.
 - The Windows PowerShell. While you can run this as either an administrator or a user without extra privileges, this procedure assumes that you are running as an administrator; to run as a user without extra privileges, use the `-verb runas` argument when you invoke the command, which is described in PowerShell documentation.
2. Go to the bin directory for your InterSystems IRIS installation.
3. In that directory, invoke InterSystems IRIS at the command line using the appropriate switch and passing in the username and password for the emergency user. This depends on the command prompt that you are using:
 - For the Windows Command prompt, the command is:

```
iris start <instance> /EmergencyId=<username>,<password>
```

This starts an emergency-mode InterSystems IRIS session with only one allowed user where:

- `<instance>` specifies the instance being started in emergency mode
- `<username>` is the sole user of the system
- `<password>` is that user's password

- For the Windows PowerShell, the command is:

```
start-process .\iris.exe -ArgumentList "start <instance> /EmergencyId=<username>,<password>"
```

This starts an emergency-mode InterSystems IRIS session with only one allowed user where:

- `<instance>` specifies the instance being started in emergency mode
- `<username>` is the sole user of the system
- `<password>` is that user's password

Note: On Windows, unlike other operating systems, the `EmergencyId` switch is preceded by a slash ("/").

For example, at the instance `MyIRIS`, to start InterSystems IRIS in emergency mode with user `jmd` with the password `purple22`, the command would be:

```
iris start MyIRIS /EmergencyId=jmd,purple22
```

The only user who can then log in is the emergency user, using the appropriate password, such as:

```
Username: jmd
Password: *****
Warning, bypassing system security, running with elevated privileges
```

Once InterSystems IRIS has started, you can start the Terminal from the InterSystems IRIS launcher or run any web application. This provides access to the Management Portal and all character-based utilities. Using this access, you can change any settings as necessary and then restart InterSystems IRIS in its normal mode.

Invoke Emergency Access Mode on UNIX®, Linux, and macOS

To start InterSystems IRIS in emergency access mode, you must either have root access or be the owner of the instance. Invoke InterSystems IRIS at the command line using the appropriate switch and passing in the username and password for the emergency user:

```
./iris start <instance-name> EmergencyId=<username>,<password>
```

This starts an emergency-mode InterSystems IRIS session with only one allowed user where:

- `<instance-name>` specifies the instance being started in emergency mode
- `<username>` is the sole user of the system
- `<password>` is `<username>`'s password

Note: If going from one of these operating systems to Windows, remember that on Windows only, the `EmergencyId` switch is preceded by a slash ("/").

For example, at the instance `MyIRIS`, to start InterSystems IRIS in emergency mode with user `jmd` with the password `purple22`, the command would be:

```
./iris start MyIRIS EmergencyId=jmd,purple22
```

The only user who can then log in is the emergency user, using the appropriate password, such as:

```
Username: jmd
Password: *****
Warning, bypassing system security, running with elevated privileges
```

Once InterSystems IRIS has started, you can run the Terminal or any web application. This provides access to the Management Portal and all character-based utilities. Using this access, you can change any settings as necessary and then restart InterSystems IRIS in its normal mode.

System Security Settings Page

The System Security Settings page (**System Administration > Security > System Security**) provides links to pages that configure the entire InterSystems IRIS® instance for security. These pages are:

- [System-Wide Security Parameters](#)
- [Authentication/Web Session Options](#)
- [LDAP Options](#)

System-Wide Security Parameters

This topic describes security issues that affect an entire InterSystems IRIS instance. This includes the system-wide security parameters and handling sensitive data in memory images.

InterSystems IRIS includes a number of system-wide security parameters. You can configure these on the **System Security Settings** page (**System Administration > Security > System Security > System-wide Security Parameters**). These are:

- **Enable audit** — Turns auditing on or off. This check box performs the same action as the **Enable Auditing** and **Disable Auditing** links on the **Auditing** page (**System Administration > Security > Auditing**). For more information on auditing, see [Auditing Guide](#). [Default is off]
- **Freeze system on audit database error** — (Only available when auditing is enabled.) Stops (freezes) the instance if there is an error writing to the audit database. For more information, see [Freezing the System If It Is Impossible to Write to the Audit Database](#).
- **Enable configuration security** — Specifies whether configuration security is on or off, as described in [Configuration Security](#). [Default is off]
- **Default security domain** — Allows you to choose the instance's default security domain. For more information on security domains, see the section [Manage InterSystems IRIS Security Domains](#). [Default is the domain established during installation]
- **Inactive limit (0–365)** — Specifies the maximum number of days that a user account can be inactive, which is defined as the amount of time between successful logins. When this limit is reached, the account is disabled. A value of 0 (zero) means that there is no limit to the number of days between logins. [Default is described in [Initial User Security Settings](#).]

Note: Mirror Members automatically set the InactiveLimit parameter to 0 on startup. This prevents user accounts from becoming inactive on other mirror members.

- **Invalid login limit (0-64)** — Specifies the maximum number of successive unsuccessful login attempts. After this limit is reached, either the account is disabled or an escalating time delay is imposed on each attempt; the action depends on the value of the **Disable account if login limit reached** field. A value of 0 (zero) means that there is no limit to the number of invalid logins. [Default is 5]
- **Disable account if login limit reached** — If checked, specifies that reaching the number of invalid logins (specified in the previous field) causes the user account to be disabled.
- **Password Expiration Days (0–99999)** — Specifies how frequently passwords expire and, therefore, how frequently users must change their passwords (in days). When initially set, specifies the number of days until passwords expire. A value

of 0 (zero) means that the password never expires; however, setting this field to 0 does not affect users for whom the **Change Password on Next Login** field has been set. [Default is 0]

CAUTION: This setting affects all accounts for the InterSystems IRIS instance, including those used by InterSystems IRIS itself. Until passwords are updated for these accounts, it may be impossible for various operations to proceed and this may lead to unexpected results.

- **Password pattern** — Specifies the acceptable format of newly created passwords. See [Password Strength and Password Policies](#) for more information. [Default is described in [Initial User Security Settings](#).]
- **Password validation routine** — Specifies a user-provided routine (or entry point) for validating a password. See the **PasswordValidationRoutine** method in the `Security.System` class for more information.
- **Role required to connect to this system** — If set to an existing role, specifies that a user must be a member of this role (as a login role) in order to log in to the system.

If you are using [LDAP authentication](#) or [OS-based LDAP authorization](#), InterSystems strongly recommends that you create a role that is required to connect and that you specify its name in this field. For more information, see [Setting Up a Role Required for Login](#).

- **Enable writing to percent globals** — Specifies whether write access to percent globals is implicitly granted to all users; if not checked, write access is controlled by normal security mechanisms. For more information on the percent globals and IRISYS (the database that holds them), see [IRISYS, the Manager's Database](#). [Default is controlled by normal security mechanisms.]
- **Allow multiple security domains** — Specifies whether there is support for multiple InterSystems security domains. For more information on security domains, see the section [Manage InterSystems IRIS Security Domains](#). [Default is a single domain]
- **Superserver SSL/TLS Support** — Specifies if the superserver supports or requires the use of TLS for client connections.

Important: Before you can configure the superserver to use TLS, there must be an existing configuration called `%SuperServer`. For more information about using TLS with the InterSystems IRIS superserver, see [Configure the InterSystems IRIS Superserver to Use TLS](#).

Options are:

- **Disabled** — The superserver refuses client connections that use TLS. (That is, it only accepts client connections that do *not* use TLS.)
 - **Enabled** — The superserver accepts client connections that use TLS but does not require them.
 - **Required** — The superserver requires client connections to use TLS.
- **Telnet server SSL/TLS Support** — Specifies if the telnet server supports or requires the use of TLS for client connections.

Important: Before you can configure the telnet server to use TLS, there must be an existing configuration called `%TELNET/SSL`. For more information about using TLS with the InterSystems IRIS telnet server, see [“Configure the InterSystems IRIS Telnet Server to use TLS.”](#)

Options are:

- **Disabled** — The telnet server refuses client connections that use TLS. (That is, it only accepts client connections that do *not* use TLS.)
- **Enabled** — The telnet server accepts client connections that use TLS but does not require them.
- **Required** — The telnet server requires client connections to use TLS.

- **Default signature hash** — Specifies the algorithm used by default to create an XML signature hash. For more information on the supported algorithms for creating hashes, see <https://www.w3.org/>.

Authentication Options

The **Authentication/Web Sessions Options** page (**System Administration > Security > System Security > Authentication/Web Options**) allows you to enable or disable authentication mechanisms for the entire InterSystems IRIS instance:

- If an authentication mechanism is disabled for the entire InterSystems IRIS instance, then it is not available for any service.
- If an authentication mechanism is enabled for the entire InterSystems IRIS instance, then it is available for all the services that support it. To enable the authentication mechanism for a particular service, use the **Edit Service** page for that property; this page is available by selecting the service from the **Services** page (**System Administration > Security > Services**).

Note: Not all services support all mechanisms.

The authentication options are:

- **Allow Unauthenticated access** — Users may connect without authenticating. (If login dialog appears, the user can leave the **Username** and **Password** fields blank and click **OK** to log in.)
- **Allow O/S authentication** — InterSystems IRIS uses the [operating system's user identity to identify the user](#); it then uses [InterSystems authorization](#).
- **Allow O/S authentication with Delegated authorization** — InterSystems IRIS uses the [operating system's user identity to identify the user](#); it then uses [delegated authorization](#).
- **Allow O/S authentication with LDAP authorization** — InterSystems IRIS uses the [operating system's user identity to identify the user](#); it then uses [LDAP authorization](#).
- **Allow Password authentication** — InterSystems IRIS uses its own native tools, called [instance authentication](#), to authenticate the user; it then uses [InterSystems authorization](#).
- **Allow Delegated authentication** — InterSystems IRIS uses [external \(delegated\) authentication system](#) by calling out to it. You can use delegated authentication with either [InterSystems authorization](#) or [delegated authorization](#).
- **Always try Delegated authentication** — InterSystems IRIS invokes [delegated authentication](#) code for users authenticating with [instance authentication](#) (also known as password authentication). If you use both delegated authentication and instance authentication and also require that ZAUTHENTICATE be called for instance authentication users, then select this option.
- **Allow Kerberos authentication** — InterSystems IRIS performs authentication using [Kerberos](#). You can use Kerberos authentication with either [InterSystems authorization](#) or [delegated authorization](#).
- **Allow LDAP authentication** — InterSystems IRIS uses [LDAP](#) (including Active Directory) to authenticate users. You can use LDAP for both authentication and [authorization](#), or you can use LDAP authentication with [InterSystems authorization](#).
- **Allow LDAP cache credentials authentication** — InterSystems IRIS uses a copy of [cached LDAP credentials](#) to authenticate LDAP users if the LDAP database becomes unavailable.
- **Allow creation of Login Cookies** — InterSystems IRIS uses cookies that are shared among enabled web applications to authenticate users, so that they do not need to enter a username and password when first using a new application. This is only relevant for web applications that use CSP.
- **Login Cookie expire time (secs)** — The duration of a login cookie, in seconds. This field is only relevant if login cookies are enabled for the instance.

- **Allow Two-factor Time-based One-time Password authentication** — InterSystems IRIS provides a [verification code via an authentication device or an app](#) that runs on the user's phone; the user then enters the code to complete the authentication process. If selected, the **Authentication/Web Session Options** page displays the fields for [configuring two-factor authentication](#).
- **Allow Two-factor SMS text authentication** — InterSystems IRIS provides a [security code via a mobile phone text message](#); the user then enters the code to complete the authentication process. If selected, the **Authentication/Web Session Options** page displays the fields for [configuring two-factor authentication](#).

If there are multiple supported [authentication](#) options, InterSystems IRIS uses [cascading authentication](#).


Effect of Changes

When you make changes to various security settings, the amount of time for these to take effect are as follows:

- Changes to user properties, such as the roles assigned to the user, are effective with the next login for that user. They have no effect on processes that are already running.
- Changes to services, such as whether a service is enabled or authentication is required, are effective for future connection attempts. Existing connections are not affected.
- Changes to role definitions are effective immediately for any subsequent privilege checks. These affect database resources immediately, because they are checked for each database access. For services and applications, they are effective with subsequent connection attempts or application initiations.

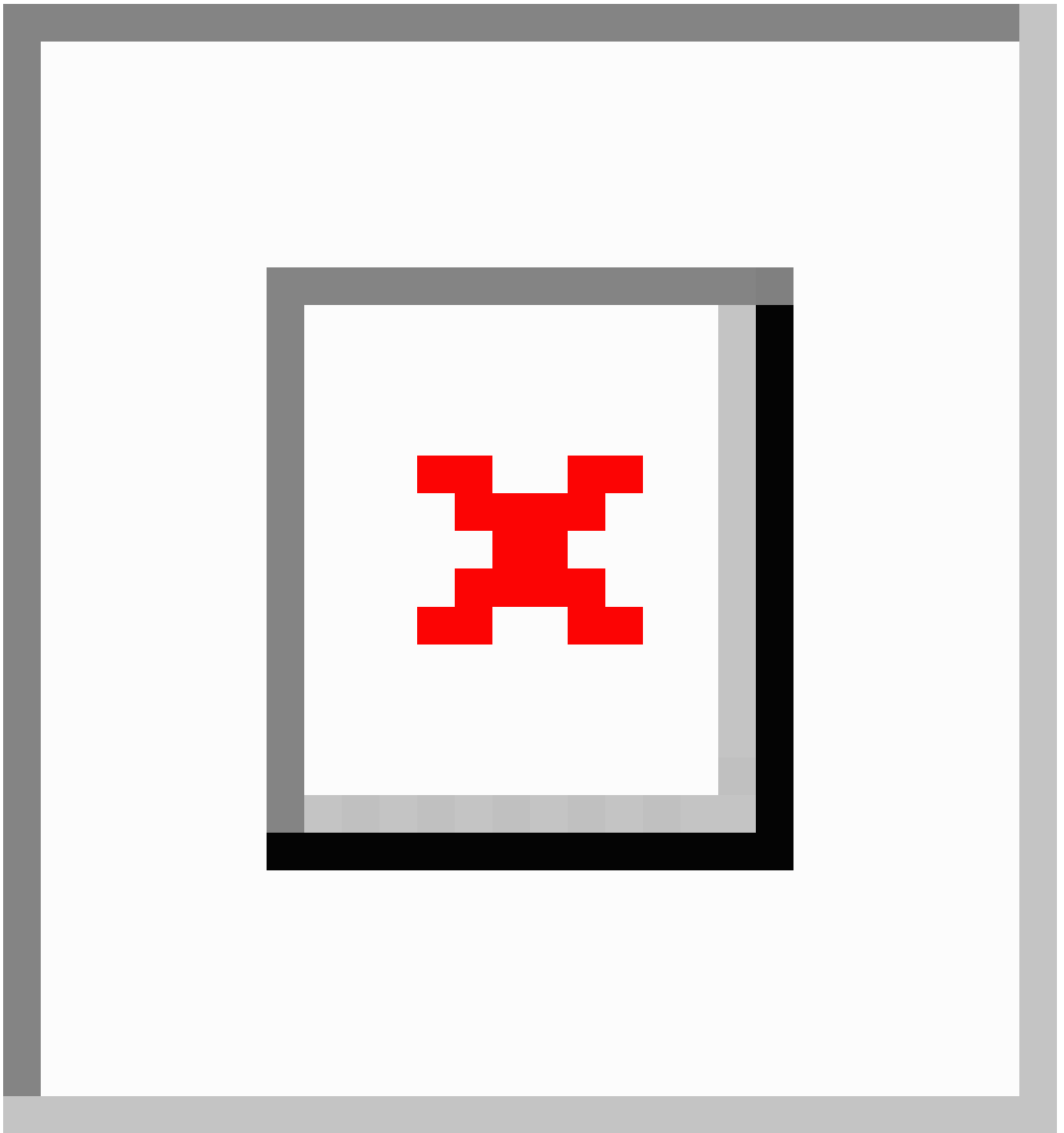
Note: The times listed here are the latest times that changes take effect; in some cases, changes may be effective earlier than indicated.

Enabling Automatic Refreshes of Management Portal Pages

By default, users can refresh Management Portal pages only by clicking the  (**Refresh the diagram**) icon where it is available. However, InterSystems IRIS enables you to provide users with a mechanism for automatically refreshing Management Portal pages every several seconds. You can modify the `^%SYS` global to expose the mechanism from the Terminal as follows:

```
set ^%SYS("Portal","EnableAutoRefresh") = 1
```

If you do so, a set of radio buttons appears on Management Portal pages that can be refreshed, enabling users to turn automatic refreshes on and off. On some pages, users can specify the refresh interval. Importantly, when you set the `EnableAutoRefresh` node to 1, automatic refreshes are off by default. The following image shows the **Classes** page when automatic refreshes have been enabled and the refresh interval has been set to 15 seconds:



Important: An automatic refresh constitutes a call to the InterSystems IRIS server and can prevent automatic logouts if they are enabled. For more information, see [Automatic Logout Behavior in the Management Portal](#).

Automatic Logout Behavior in the Management Portal

Each InterSystems IRIS Management Portal web application has a [Session Timeout](#) property that determines the length of time that users can remain inactive before their sessions expire. By default, fifteen seconds after a user's session expires, the Management Portal refreshes the current page and logs the user out. The Management Portal does not cache pending changes or prompt the user to save pending changes. Unsaved changes are discarded.

Important: Inactivity is the time between calls to the InterSystems IRIS server. Not all user actions constitute a call to the server. For example, a user clicking **Save** constitutes a call to the server, but a user typing in a text field does not. Consequently, if a user is editing a data transformation, but does not click **Save** for longer than **Session Timeout** threshold, then the user's session expires and any unsaved changes are discarded.

After an automatic logout, the following scenarios may occur:

- The login page appears.
- The Management Portal logs the user out and then immediately logs the user in again because the web application has a **Group By Id** value that results in automatic authentication. In this case, the current Management Portal page appears refreshed and any pending changes are removed.

You can take the following steps to prevent users from losing work:

- Remind users to save their work on a regular basis.
- Extend the **Session Timeout** value for web applications where users are performing time-intensive configuration tasks such as modifying data transformations. The default **Session Timeout** value is 15 minutes.

Additionally, while InterSystems recommends that you retain the default automatic logout behavior, you can allow users to remain logged in until they actively log out or close their browsers when they are on **Interoperability** pages in the Management Portal. To do so, use *^EnsPortal* as follows:

```
^EnsPortal("DisableInactivityTimeout","Portal") = 1
```

Note: This is a per-namespace setting. To modify logout behavior, you must set this value for each namespace individually.

You can reinstate automatic logouts by using *^EnsPortal* again:

```
^EnsPortal("DisableInactivityTimeout","Portal") = 0
```

InterSystems recommends that you consider the possible security implications before you make any changes.

For more information about web applications and their settings, see [Defining Applications](#).

Other Security Features

This section describes several additional security features and considerations. These are:

- [Enable Use of the Secure Debug Shell](#)
- [Protect Sensitive Data in Memory Images](#)

The following cover additional security topics:

- [Protecting InterSystems IRIS Configuration Information](#)

Enable Use of the Secure Debug Shell

InterSystems IRIS includes the ability to suspend a routine and enter a shell that supports full debugging capabilities (as described in [Command-line Routine Debugging](#)). InterSystems IRIS also includes a [secure debug shell](#), which has the advantage of ensuring that users are prevented from exceeding or circumventing their assigned privileges.

By default, users at the debug prompt maintain their current level of privileges. To enable the secure shell for the debug prompt and thereby restrict the commands that the user may issue, the user must hold the **%Secure_Break:Use** privilege (the **Use** permission for the **%Secure_Break** resource). To give a user this privilege, make the user a member of a role which includes the **%Secure_Break:Use** privilege, such as the predefined **%SecureBreak** role.

Protect Sensitive Data in Memory Images

Certain error conditions can cause the contents of a process's memory to be written to a disk file, known as a "core dump." This file contains copies of all data that was in use by the process at the time of the dump, including potentially sensitive application and system data. This can be prevented by disallowing core dumps on a system-wide basis. The method for disallowing core dumps varies according to the operating system in use; for details, consult the documentation of your operating system.

Tighten Security for an Instance

Tighten Security for an Instance

To provide increased security for an InterSystems IRIS® database, you can and should configure it to more tightly constrain user access. This can prevent unauthorized users from using system tools or from gaining access to sensitive resources. This section describes various actions that reduce the attack surface of a database instance or otherwise increase its security. The section assumes you have already installed an InterSystems IRIS instance with an initial security of Minimal. If you have chosen an initial security of Normal or Locked Down, some of these actions have already been performed for you.

There actions for tightening an instance's security are presented here roughly in the sequence in which they should be performed:

- [Enable auditing](#)
- [Change the authentication mechanism for an application](#)
- [Restrict access to services](#). This involves:
 - [Limit the number of enabled services](#)
 - [Limit the number of public services](#)
 - [Restrict access to services by IP address or machine name](#)
- [Limit remote privileged access](#)
- [Limit the number of privileged users](#)
- [Disable the _SYSTEM user](#)
- [Restrict access for UnknownUser](#)
- [Configure third-party software](#)

The InterSystems [Security Advisor](#) also provides an automated analysis of the instance and recommendations for actions to increase the security of an instance.

Important: An InterSystems IRIS database instance has many interdependent elements. Because of this, it is recommended that you only do what is specified for a change, and not more or less. For example, simply removing UnknownUser from the %A11 role — without doing anything else — will cause problems for a minimal-security installation.

Enable Auditing

The primary elements of security are often described as the “Three A’s”: authentication, authorization, and auditing. [Auditing](#) provides two functions:

- It provides data about what has occurred if there is a security event.
- The knowledge of its existence can be a deterrent for an attacker, given that the attack will be tracked and there will be evidence of any malicious actions.

To enable auditing for key events, the procedure is:

1. From the Management Portal home page, select **System Administration > Security > Auditing > Enable Auditing**. If the choice is not available, auditing is already enabled.
2. From the Management Portal home page, go to **Configure System Events** page (**System Administration > Security > Auditing > Configure System Events**).

3. On the **Configure System Events** page, enable the following events if they are not already enabled by clicking **Change Status** in the event's row:
 - `%System/%DirectMode/DirectMode` — Provides information on console/terminal use. For sites that extensively use command-line utilities, can create large amounts of data. Recommended if increased data is not an issue.
 - `%System/%Login/Login` — Provides information on logins. For large sites, can create large amounts of data. Recommended if increased data is not an issue.
 - `%System/%Login/LoginFailure` — Provides feedback on possible attempted unauthorized logins. Recommended.
 - `%System/%Security/Protect` — Provides data on attempts to read, write, or use protected data. Recommended.

Change the Authentication Mechanism for an Application

A key element of restricting access to the database is configuring the instance to use a stricter authentication mechanism for its applications. This section describes how to perform this procedure, using the Management Portal as an example application and with the change from unauthenticated access (as in a minimal-security installation) to requiring a password as an example of moving to a stricter authentication mechanism.

- Important:** Performing the following procedure may affect aspects of the instance being modified beyond access to the Portal. The specifics depend on (1) the instance's configuration and (2) whether you are performing just this procedure or all the procedures in this section. Specifically:
- [Making `%Service_WebGateway:Use not public`](#) means that all users of web applications will need to be granted `%Service_WebGateway:Use` by some other means.
 - [Removing `UnknownUser` from the `%All` role](#) can have many effects.

To provide properly functioning authentication for an application, there must be consistent authentication mechanisms for both the application and any service that it uses. For a web application, the Web Gateway must also be configured to match the Web Gateway service. Hence, to provide authentication for the Management Portal, there are three layers that all need to work together:

- The `%Service_WebGateway` service
- The Web Gateway
- The Management Portal application

If these layers do not have matching authentication mechanisms, this usually results in a denial of access — for example, there may be a “This page cannot be displayed” error instead of a login page or access to the Management Portal.

- Important:** If (1) a web application uses a more powerful authentication mechanism than the Web Gateway and `%Service_WebGateway` and (2) authentication succeeds, then the system's security is only that of the less powerful mechanism.

For an instance with a minimal-security installation, the Web Gateway, `%Service_WebGateway`, and the Management Portal application are all set up for unauthenticated access. To provide password-level authentication for the Portal, various InterSystems IRIS elements must be configured as follows:

- The Web Gateway service must require password authentication.
- The Web Gateway must provide a username and password for that authentication.
- The user representing the Gateway must have sufficient privilege to use the Web Gateway service.
- The Management Portal must require password authentication.
- All the Portal's users must have sufficient privilege to use the Portal.

Important: Complete the following set of procedures during a single session in the Portal. Otherwise, you may lock yourself out of the Portal and have to perform the remaining procedures through the ^SECURITY routine.

An overview of the procedure to make these changes is:

1. Optionally, [turn on auditing to track the changes to the instance](#). This is described in [Enable Auditing](#).
2. [Give the %Service_WebGateway:Use privilege to the CSPSystem user](#).
3. [Change the password of the CSPSystem user](#).
4. [Configure the Web Gateway to provide a username and password for authentication](#).
5. [Configure %Service_WebGateway to require password authentication](#).
6. [Remove the public status of the %Service_WebGateway:Use privilege](#).
7. [Configure the Management Portal application to require password authentication only](#).
8. [Specifying the appropriate privilege level for the instance's users](#).
9. Optionally, [make the class reference available](#).
10. [Begin enforcement of the new policies](#).

Once this process is complete, then a user's next attempt to connect to the Portal will result in a login prompt.

CAUTION: An InterSystems IRIS database instance has many interdependent elements. Because of this, it is recommended that you only do what is specified for a change, and not more or less. Otherwise, you may lock yourself out of the instance or could even render the instance temporarily inoperative.

Give the %Service_WebGateway:Use Privilege to the CSPSystem User

The InterSystems IRIS installation process creates a CSPSystem user, which represents the Web Gateway in its interactions with the %Service_WebGateway service. Since the service is going to have restricted access, this user needs to hold the %Service_WebGateway:Use privilege for the authentication process.

Note: There is a *service* called %Service_WebGateway and a *resource* called %Service_WebGateway. The resource regulates access to the service. Therefore, to gain access to the service, a user must have Use permission for the resource — that is, the %Service_WebGateway:Use privilege.

To associate the %Service_WebGateway:Use privilege with the CSPSystem user, the procedure is:

1. From the Management Portal home page, go to the **Roles** page (**System Administration > Security > Roles**).
2. On the **Roles** page, click **Create New Role**. This displays the **Edit Role** page, where the **Name** field is editable.
3. Enter a name for the role to include the %Service_WebGateway:Use privilege (such as “GatewayRole”).
4. Click **Save**. InterSystems IRIS has now created the role.
5. In the **Privileges** section on the **General** tab of the **Edit Role** page, click **Add**, which displays a list of available resources for the role.
6. From this list, click %Service_WebGateway and then click **Save**. The newly created role now includes the %Service_WebGateway:Use privilege.
7. Select the **Members** tab of the **Edit Role** page.
8. On this tab, you can assign the CSPSystem user to the newly created role. Click CSPSystem from the users in the **Available** list and move it to the **Selected** by clicking the right arrow.
9. Click **Assign** to assign CSPSystem to the role. (In other words, CSPSystem is now a member of the role.) This means that CSPSystem holds the %Service_WebGateway:Use privilege.

Note: The system creates the CSPSystem user to represent the Web Gateway. If you prefer, a different user can perform this function. This procedure refers only to the CSPSystem user; if you use a different user, replace CSPSystem with that username where relevant.

Change the Password of the CSPSystem User

Because a minimal-security installation gives the CSPSystem user a password of “SYS”, it is important to change this to a new password — one that an attacker would not know or be able to guess. The procedure is:

1. In the Management Portal, go to the **Users** page (**System Administration > Security > Users**).
2. On the **Users** page, click **CSPSystem**. This displays the **Edit User** page.
3. Enter the new password for CSPSystem in the **Password** field. Since no user has to remember this password, you can make it as long and complex as you wish. You will need to remember it long enough to complete the next item, [Configure the Web Gateway to Provide a Username and Password](#).
4. Reenter the new password in the **Password (Confirm)** field and click **Save**. If the Portal does not display an error message or dialog, then the password change has succeeded.

If you wish, you can also confirm that CSPSystem is assigned to the role created for authentication in the previous procedure. To do this, click on the **Roles** tab. The table with the column heading **CSPSystem is Assigned to the Following Roles** should list the newly-created role.

Configure the Web Gateway to Provide a Username and Password

Because you are going to configure **%Service_WebGateway** to require password authentication, the Web Gateway needs to provide a username-password pair. Having set up a user with the appropriate level of privilege, you have established a username-password pair that the Gateway can provide. The next step is to configure the Gateway to provide this username-password pair when the InterSystems IRIS server challenges it for them. The procedure is:

1. In the Management Portal, go to the **Web Gateway Management** page (**System Administration > Configuration > Web Gateway Management**).
2. On the **Web Gateway Management** page, select **Server Access** from the list on the left side. This displays the **Server Access** frame.
3. In the **Server Access** frame, the LOCAL server should be highlighted. Click **Submit** to edit it, which displays a page with Server Access and Error Pages parameters.
4. On this page, there is a **Connection Security** section.
5. Ensure that the **Connection Security Level** drop-down has “Password” displayed.
6. In the **User Name** field, enter CSPSystem.
7. In the **Password** and **Password (Confirm)** field, enter the password that you selected in the previous section.
8. Click **Save Configuration** near the bottom of the page.
9. To return to the Management Portal, click **Back to Management Portal** from the bottom of the list in the left pane.

Configure %Service_WebGateway to Require Password Authentication

Now that the Gateway is configured to provide a username and password and you have given the CSPSystem user the necessary level of privilege, the next step is to configure the service that manages web applications (**%Service_WebGateway**) so that it requires password authentication. The procedure is:

1. From the Management Portal home page, go to the **Services** page (**System Administration > Security > Services**).
2. On the **Services** page, click **%Service_WebGateway**. This displays the **Edit Service** page for **%Service_WebGateway**.

3. On the **Edit Service** page, under **Allowed Authentication Methods**, make sure that **Unauthenticated** access is disabled and that **Password** access is enabled (also known as “Instance Authentication”). Click **Save**.

Remove the Public Status of the %Service_WebGateway:Use Privilege

With **%Service_WebGateway** requiring password authentication and the Gateway able to authenticate with an appropriately authorized user, the next step is to exclude **%Service_WebGateway:Use** from public availability. The procedure is:

1. From the Management Portal home page, go to the **Resources** page (**System Administration > Security > Resources**).
2. On the **Resources** page, in the row for **%Service_WebGateway**, click **Edit**. This displays the **Edit Resource** page for **%Service_WebGateway**.
3. In the **Public Permission** section, clear the **Use** box. Click **Save**.

Important: Once **%Service_WebGateway:Use** is not a public privilege, only those users who have been explicitly granted it will be able to use web applications. You may need to assemble a list of these users and grant them this privilege through other means.

Configure the Management Portal to Accept Password Authentication Only

Once the connection between the Gateway and the InterSystems IRIS server has a new authentication mechanism, the next task is to configure the Management Portal application to use a matching mechanism. In this example, this mechanism is Instance Authentication. The procedure for changing the Portal’s authentication mechanism is:

1. From the Management Portal home page, go to the **Web Applications** page (**System Administration > Security > Applications > Web Applications**).
2. On the **Web Applications** page, the **/csp/sys** application represents the Management Portal home page. Click the name **/csp/sys** in this row to edit the application. This displays the **Edit Web Application** page for the **/csp/sys** application.
3. In the **Security Settings** section, under **Allowed Authentication Methods**, disable **Unauthenticated** access and enable **Password** access. Click **Save**.
4. Also disable **Unauthenticated** access and enable **Password** access for all the applications that compose the other pages and choices of the Portal. These applications are:
 - **/csp/sys/exp**
 - **/csp/sys/mgr**
 - **/csp/sys/op**
 - **/csp/sys/sec**

Note: After editing the application **/csp/sys/op**, you will need to authenticate to make further changes.

This configures the Portal to require password authentication (also known as “Instance Authentication”) and not to allow unauthenticated access, and so that all its parts behave consistently. The next step is to ensure that all relevant users have appropriate access to the Portal.

Specify the Appropriate Privilege Level for the Instance’s Users

When the Portal is configured to accept unauthenticated connections, any user can connect as the **UnknownUser**. Because a minimal-security installation makes **UnknownUser** a member of the **%A11** role, there is no danger of being locked out of the Portal. Now that the Portal requires password authentication, its legitimate users need to be members of the **%Operator** role, the **%Manager** role, or the **%A11** role.

In a minimal-security installation, SuperUser, Admin, _SYSTEM, and UnknownUser all have this level of privilege; further, these all have passwords of “SYS”.

Note: In a normal or locked-down installation, the UnknownUser is enabled, but is not assigned any roles.

In a normal or locked-down installation, passwords are set in the installation process, but you can choose to change them again here.

To properly secure users, the procedure is:

1. Either disable UnknownUser or remove UnknownUser from the %All role.
 - To disable UnknownUser, the procedure is:
 - a. On the Users page (**System Administration > Security > Users**), click **UnknownUser** under the **Name** column. This displays the **Edit User** page for UnknownUser.
 - b. Clear the **User Enabled** field and click **Save**.
 - To remove UnknownUser from the %All role:
 - a. On the **Users** page (**System Administration > Security > Users**), click **UnknownUser** under the **Name** column. This displays the **Edit User** page for UnknownUser.
 - b. Go to the **Roles** tab on the **Edit User** page.
 - c. In the **User UnknownUser is Assigned to the Following Roles** table, on the %All row, and click **Remove**.

Important: Limiting access through UnknownUser can have widespread effects, particularly if an instance’s users are not sufficiently privileged.

2. Ensure that any other potentially unauthorized users are not members of %All, %Developer, %Manager, %Operator, %SQL, or any user-defined role that grants privileges. This involves a process analogous to removing UnknownUser from the %All role.

(A user-defined role that grants privileges might have Use permission on any of the %Admin... resources, %Development, or any of the %Service or %System resources, or Write permission on %DB_IRISLIB or %DB_IRISSYS.)

3. Ensure that any user who *should* have access to the Portal is assigned to %All, %Developer, %Manager, %Operator, %SQL, or any user-defined role that grants Portal access. The procedure, for each of these users, is:
 - a. On the **Users** page (**System Administration > Security > Users**), click the name of the user under the **Name** column. This displays the **Edit User** page for that user.
 - b. Go to the **Roles** tab on the **Edit User** page.
 - c. Move the desired role(s) from the **Available** to the **Selected** list by selecting the role, clicking the right arrow button, and then clicking **Assign** to assign the user to the role(s).
4. Change the passwords for SuperUser and Admin users from the default and disable the accounts. To do this:
 - a. On the **Users** page (**System Administration > Security > Users**), click the name of the user under the **Name** column. This displays the **Edit User** page for that user.
 - b. Click **Enter new password**.
 - c. Enter the new password in the **Password** field.
 - d. Confirm it in the **Password (confirm)** field.
 - e. Clear the selection for **User enabled** and click **Save**.

Note: InterSystems IRIS requires at least one enabled account with the **%A11** role. InterSystems recommends creating a unique user with the **%A11** role and disabling the SuperUser, Admin, and _SYSTEM users.

Important: Make sure that you know the password for at least one user who administers the Portal. Otherwise, you may lock yourself out of the Portal and have to log in using [emergency access](#) so that you can reset one or more passwords using the **^SECURITY** routine.

Make the Class Documentation Available

Once you finish configuring the service, the Web Gateway, and the Portal application, you may wish to ensure that the class documentation program is available. The procedure is:

1. From the Management Portal home page, go to the **Web Applications** page (**System Administration > Security > Applications > Web Applications**).
2. To make the documentation available:
 - a. On the **Web Applications** page, the `/csp/documatic` application represents the class reference application. Click `/csp/documatic` in this row to edit the application. This displays the **Edit Web Application** page for the `/csp/documatic` application.
 - b. In the **Security Settings** section, under **Allowed Authentication Methods**, disable **Unauthenticated** access and enable **Password** access. Click **Save**.

Note: In a normal installation, password access is already enabled.

If you do not perform this procedure, the service requires a password prompt but the application attempts to use unauthenticated access. This prevents all users — including those assigned to **%All** — from reaching the documentation.

Begin Enforcement of New Policies

At this point, the InterSystems IRIS instance is fully configured to operate properly. However, all existing connections are still using unauthenticated access. To begin enforcement of the new policies, the following events must occur:

- [The Web Gateway must establish an authenticated connection.](#)
- [All users must also establish authenticated connections.](#)

Establish an Authenticated Web Gateway Connection

To force the Web Gateway to establish an authenticated connection, the procedure is:

1. From the Management Portal home page, select **System Administration > Configuration > Web Gateway Management**. This displays the **Web Gateway Management** page.
2. On the **Web Gateway Management** page, select **Close Connections** from the list on the left side. This displays the **Close Connections** frame.
3. Click **Close Connection(s)**. This displays a message indicating that all connections between the Gateway and InterSystems IRIS server have been closed.

The next time that a user requests a page, the Gateway will reestablish a connection to the InterSystems IRIS server. This connection will use the selected authentication mechanism.

Establish Authenticated User Connections

At this point, all connections to the Management Portal are still using unauthenticated access. If there is no pressing need to require authenticated access, then there is nothing else to do. Users will gradually end their connections to the Portal and

will have to authenticate when they reconnect. (Connections may be ended due to machine reboots, stopping and restarting browsers, clearing browser caches, Portal logouts, etc.)

If there is a need to force connections to use authenticated access, you can stop and restart InterSystems IRIS. For example, on Windows, if you have InterSystems IRIS available through the default Start menu page:

1. From the Windows Start menu, select **Programs > InterSystems IRIS**, then the InterSystems IRIS instance to restart.
2. On the submenu for the instance of InterSystems IRIS, choose **Stop InterSystems**.
3. On the dialog that appears, select **Restart** and click **OK**.

Note: If you are using a production instance of InterSystems IRIS, you may want to choose a low-traffic time for the restart, since users will temporarily not have access to either InterSystems IRIS as a whole or the Portal.

Limit the Number of Public Resources

Any [resource](#) can be specified as a public resource. This means that any user has the ability to read, write, or use the resource, depending on its public settings. The following should always be public:

Table A-2: Required Public Resources and Their Permissions

Resource	Permission
%DB_IRISLOCALDATA	R
%DB_IRISLIB	R
%DB_IRISTEMP	RW

To tighten the security of an instance, limit the number of public resources. To do this, the procedure is:

1. Ensure that all users who genuinely require access to these resources have been given privileges for them.

Important: If you do not provide privileges for %Service_WebGateway:Use to the appropriate users, then this procedure can result in a widespread lockout from the Management Portal and other web applications.

2. From the Management Portal home page, go to the **Resources** page (**System Administration > Security > Resources**).
3. On the **Resources** page, each resource for which there is one or more public permissions has those permissions listed in the **Public Permissions** column of the table of resources. Select the resource by clicking **Edit**. This displays the resource's **Edit Resource** page.
4. On the **Edit Resource** page, clear any checked **Public Permission** fields and click **Save**. The resource is no longer public.

Perform this procedure for all public resources.

Restrict Access to Services

There are various pathways by which users can interact with InterSystems IRIS. [Services](#) regulate access to these pathways. To limit access to InterSystems services, the available choices are:

- [Limit the number of enabled services](#) to only those required for the applications in use
- [Limit the number of public services](#) to only those required for the applications in use
- [Restrict access to services by IP address or machine name](#)

Limit the Number of Enabled Services

To limit the number of enabled services, the procedure is:

1. Determine the required services for the InterSystems IRIS instance. Typically, these are:
 - Whatever service is required for each form of user access
 - Whatever services are required for any automated access
 - Either `%Service_Console` (on Windows) or `%Service_Terminal` (on UNIX®), for local programmer-mode access
2. From the Management Portal home page, go to the **Services** page (**System Administration** > **Security** > **Services**).
3. On the **Services** page, for each service that is not required, select the service by clicking on its name. This displays the service's **Edit Service** page.
4. On the **Edit Service** page, clear the **Service Enabled** field and click **Save**. The service is now disabled.

Once you have disabled all unnecessary services, the only pathways to InterSystems IRIS are the required services.

Limit the Number of Public Services

Each service is associated with a resource. In most cases, the resource has the same name as the service, such as `%Service_WebGateway`; the exception to this is the `%Service_Bindings` service, which is associated with the `%Service_Object` and `%Service_SQL` resources. Services are public because of the settings for the resources associated with them. Because of this, the procedure for making a service non-public is the same as for making any other resource non-public. This is described in [Limiting the Number of Public Resources](#).

Restrict Access to Services by IP Address or Machine Name

For certain services, you have the option of restricting access to the service according to IP address or machine name. This is known as the ability to limit “allowed incoming connections.” The services that support this feature are:

- `%Service_Bindings`
- `%Service_CacheDirect`
- `%Service_ECP`
- `%Service_Monitor`
- `%Service_Shadow`
- `%Service_WebGateway`

By default, a service accepts connections from all machines. If a service has no associated addresses or machine names, then it accepts connections from any machine. If one or more addresses or machine names are specified from which a service accepts connections, then the service only accepts connections from these machines.

This feature is not available for `%Service_CallIn`, `%Service_ComPort`, `%Service_Console`, `%Service_DataCheck`, `%Service_Login`, `%Service_Mirror`, `%Service_Telnet`, and `%Service_Terminal`.

To restrict access to a service by IP address, the procedure is:

1. Determine the IP addresses of those machines with legitimate access to the service.
2. From the Management Portal home page, go to the **Services** page (**System Administration** > **Security** > **Services**).
3. On the **Services** page, for each service for which you are restricting access by IP address, select the service by clicking on its name. This displays the service's **Edit Service** page.
4. On the **Edit Service** page, in the Allowed Incoming Connections section, click **Add New**.

5. In the displayed dialog, enter an IP address for an allowed incoming connection. Click **OK**.
6. Click **Add New** and enter other addresses as needed.

Perform this procedure for each service that is restricting the IP addresses from which it accepts connections.

Limit Remote Privileged Access

InterSystems IRIS supports ECP remote job requests. However, a remote job runs as root on the server, which could allow a user to operate on the server with more privileges than intended. To disable handling of remote jobs and limit remote privileged access on the server, follow the procedure in [Changing This Parameter](#) to set `net job` to `false`. This setting is `true` by default.

Limit the Number of Privileged Users

Every instance of InterSystems IRIS must have at least one user who is assigned to the `%All` role. In fact, if there is only one user assigned to this role, then InterSystems IRIS prevents the user from being removed from the role. However, over time, an instance can end up having more users assigned to `%All` than are necessary. This can arise from assigned users leaving an organization but their accounts not being disabled, from temporary assignments not being revoked, and so on.

Along with the `%All` role, the system-defined roles of `%Manager`, `%Developer`, `%Operator`, and `%SQL` can give users undue privilege. There also may be user-defined roles that do this. Users assigned to such roles are sometimes known as “privileged users.”

To limit the number of privileged users, determine which users are assigned to each privileged role and remove those who are not needed. The procedure is:

1. From the Management Portal home page, go to the **Roles** page (**System Administration** > **Security** > **Roles**).
2. On the **Roles** page, click the name of the role. This displays the **Edit Role** page for that role.
3. On the **Edit Role** page, click the **Members** tab, which displays a list of the users and roles assigned to the role.
4. To remove any user from the specified role, click **Remove** on the row for the user or role to be removed.

Perform this procedure for each privileged role, including `%All` and the others listed previously. It is also important to disable the `_SYSTEM` user; the procedure for this is described in [Disabling the _SYSTEM User](#).

Important: Certain seemingly non-privileged roles may have what could be called “privileges by proxy.” This occurs when a seemingly non-privileged role is assigned to a privileged role. In this case, any user who is assigned to role with privileges by proxy holds all the privileges associated with the privileged role.

Avoid creating privileges by proxy whenever possible. When not possible, have as few users as possible assigned to the roles with privileges by proxy.

Disable the _SYSTEM User

The InterSystems IRIS installation program creates the `_SYSTEM` user. This user is created in accordance with the SQL standard as the SQL root user. In a minimal-security installation, the default password for this user is “SYS”; in normal and locked-down installations, the default password is whatever was selected during the installation process. Because this user and the password of “SYS” are both publicly specified by the SQL standard, and because of this user’s SQL privileges, disabling `_SYSTEM` is important for tightening access to an InterSystems IRIS instance.

To do this, the procedure is:

1. From the Management Portal home page, go to the **Users** page (**System Administration** > **Security** > **Users**).
2. On the **Users** page, click the name `_SYSTEM` to open the **Edit User** page for `_SYSTEM`.
3. On the **Edit User** page for `_SYSTEM`, clear the **User Enabled** field. Click **Save**.

Note: If you need to check root-level SQL privileges after disabling `_SYSTEM`, you will have to temporarily enable the user to perform the required action.

Restrict Access for UnknownUser

In instances that support [unauthenticated access](#), connections that do not use authentication are established with the [UnknownUser](#) account. In minimal-security installations, the default behavior is that:

- All connections use UnknownUser.
- UnknownUser is assigned to the `%A11` role.
- UnknownUser holds all SQL privileges.

To restrict access for UnknownUser, disable unauthenticated access for all enabled services. (Other actions may not be effective or may result in a [lockout](#) from the Management Portal.)

Note: If you have performed all of the previous actions listed in this section, you may have already disabled UnknownUser and limited the number of public resources.

Potential Lockout Issue with the UnknownUser Account

If an instance has been installed with Minimal security, UnknownUser has the `%A11` role; the instance also has unauthenticated access available for all services and applications. If you simply change the user's role (from `%A11` to something else) and still allow unauthenticated access, you may be unable to use basic features.

This is because, under these conditions, InterSystems IRIS establishes a connection to the selected tool without performing authentication. When there is no authentication, the system automatically sets the user account to UnknownUser. The next step is to check user privileges. If UnknownUser has insufficient privileges, access to the tool is limited or not available. For example, under these circumstances, the Terminal displays an “Access Denied” message and then shuts down; the Portal displays its main page, but no options can be selected.

To correct this condition:

1. Start InterSystems IRIS in [emergency access mode](#).
2. Restore sufficient privileges to the UnknownUser account.

If you wish to prevent use of UnknownUser, you must [upgrade the authentication mechanism for the Management Portal](#).

Configure Third-Party Software

InterSystems products often run alongside and interact with non-InterSystems tools, including virus scanners. For important information about the effects these interactions can have, see [Configuring Third-Party Software to Work in Conjunction with InterSystems Products](#).

Security Advisor

Security Advisor

To assist system managers in securing an InterSystems IRIS system, the InterSystems IRIS Management Portal includes a tool called the Security Advisor. This is a web page that shows current information related to security in the system configuration. It recommends changes or areas for review, and provides links to other pages in the Management Portal so that you can make the recommended changes.

Important: The Security Advisor provides general recommendations, but does not have any knowledge of an instance's needs or requirements. It is important to remember that each InterSystems IRIS instance has its own requirements and constraints, so that issues listed in the Security Advisor may not be relevant for your instance; at the same time, the Security Advisor may not list issues that are of high importance for you. For example, the Security Advisor exclusively recommends that services use Kerberos authentication, but, depending on your circumstances, authentication through the operating system, Instance Authentication, or even unauthenticated access may be appropriate.

There are some general features in the Security Advisor:

- **Details** button — Each section has a **Details** button. Selecting this button displays the page for managing that aspect of InterSystems IRIS regulated by the section.
- **Name** button — Each named item in each section is a link. Selecting one of these items displays the page for managing that item.
- **Ignore** check box — Each named item in each section has an associated **Ignore** check box. If you have determined that the item does not apply to your specific requirements, selecting this box grays out the line for the specified item. The line does not appear if InterSystems IRIS is set up according to the Security Advisor's recommendations, whether or not the **Ignore** check box is selected.

Auditing

This section displays recommendations on auditing itself and on particular audit events:

- Auditing should be enabled — Auditing creates a record that can provide forensic information after any notable or unusual system events.
- Auditing for this event type should be enabled — Auditing particular events can provide more specific information about various topics. Specifically, since the events noted when not enabled are:
 - The DirectMode event — Auditing this event can provide information about connections to InterSystems IRIS that give users significant privileges.
 - The Login event — Auditing this event can provide information questionable logins.
 - The LoginFailure event — Auditing this event can provide information about attempts to gain inappropriate access to the system.

Services

This section displays recommendations on InterSystems services. For each service, depending on its settings, the Security Advisor may address any of the following issues:

- Ability to set % globals should be turned off — Since % globals often hold system information, allowing users to manipulate these globals can result in serious, pervasive, and unpredictable effects.

- Unauthenticated should be off — Unauthenticated connections give all users, including the unidentified **UnknownUser** account, unregulated access to InterSystems IRIS through the service.
- Service should be disabled unless required — Access through any service monitored by the Security Advisor can provide an undue level of system access.
- Service should use Kerberos authentication — Access through any other authentication mechanism does not provide the maximum level of security protection.
- Service should have client IP addresses assigned — By limiting the number of IP addresses from which connections are accepted, InterSystems IRIS may be able to more tightly oversee the connections to it.
- Service is Public — Public services give all users, including the unidentified **UnknownUser** account, unregulated access to InterSystems IRIS through the service.

Roles

This section displays recommendations for all roles that hold possibly undue privileges; other roles are not listed. For each role, the Security Advisor may address any of the following issues:

- This role holds privileges on the Audit database — Read access to the Audit database may expose audited data inappropriately; Write access to the Audit database may allow the inappropriate insertion of data into that database.
- This role holds the **%Admin_Secure** privilege — This privilege can allow for the establishing, modifying, and denying access of users to assets; it also allows the modification of other security-related features.
- This role holds Write privilege on the %IRISSYS database — Write access to the %IRISSYS database may allow the compromise of system code and data.

Users

This section displays recommendations related to users generally and for individual user accounts. In this area, the Security Advisor may address any of the following issues:

- At least 2 and at most 5 users should have the **%All** role — Too few users holding **%All** can lead to access problems in an emergency; too many users holding it can open the system to compromise
- This user holds the %All role — Explicitly announcing which users hold **%All** can help eliminate any who hold it unnecessarily.
- UnknownUser account should not have the **%All** role — A system cannot be properly secured if anonymous users have all privileges. While this is part of any instance with a Minimal security level, such an instance is not properly secured by design.
- Account has never been used — Unused accounts provide an attractive point of entry for those attempting to gain unauthorized access.
- Account appears dormant and should be disabled — Dormant accounts (those that have not been used for over 30 days) provide an attractive point of entry for those attempting to gain unauthorized access.
- Password should be changed from default password — This is a commonly attempted point of entry for those attempting to gain unauthorized access.

Web, Privileged Routine, and Client Applications

Each application type has its own section, which makes it simpler to review details for each application type. These sections display recommendations related to access to and privileges granted by applications. In this area, the Security Advisor notes the following issues:

- Application is Public — Public applications give all users, including the unidentified **UnknownUser** account, unregulated access to the data associated with the application and actions that the application supports. This is even more notable if the application also grants the **%All** role, either conditionally or absolutely.
- Application conditionally grants the **%All** role — A system cannot be properly secured if users have the possibility of holding all privileges. This is even more notable if the application is also public.
- Application grants the **%All** role — A system cannot be properly secured if users have all privileges. This is even more notable if the application is also public.

Secure InterSystems Processes and Operating-System Resources

Introduction

This document describes how to reduce the potential attack surface available to an intruder by hardening the operating system on which an instance of an InterSystems IRIS® data platform runs. Topics include:

- The operating system services that an InterSystems IRIS instance requires
- The various types of InterSystems IRIS processes, along with the purpose of each
- Methods for identifying the function of InterSystems IRIS processes in a running instance
- How to remove or disable optional InterSystems IRIS processes that your site may not need
- Processes required by an instance that is running, in addition to either the iris process on UNIX® or the irisdb.exe process on Windows
- The TCP and UDP ports for InterSystems IRIS processes, along with the purpose of each

InterSystems IRIS Processes

Most processes comprising an InterSystems IRIS instance use the iris executable on UNIX® and the irisdb.exe executable on Windows, and each of these files resides in the bin directory under the installation directory. A running instance uses a number of system processes to coordinate and support the processes running user code. InterSystems IRIS processes can be examined in the Management Portal under **System Operation > Processes**.

Core Processes

Core system processes are started early in the initialization of an instance and have no value in the **User** column. You can identify these processes by the value in the **Routine** column which, in the case of system processes, does not always contain the name of an InterSystems IRIS routine. The **Routine** column lists the following core system processes by name:

- **CONTROL** — Creates and initializes shared memory and provides various control functions.
- **WRTDMN** — Performs all writes to databases and WIJ (the write daemon).
- **GARCOL** — Garbage collects large kills.
- **JRNDMN** — Performs journal writes.
- **EXPDMN** — Performs database expansions.
- **AUXWD** — Performs write daemon tasks (write daemon auxiliary workers).
- **MONITOR** — Writes alerts to the alert file and transmits email alerts.
- **CLNDMN** — Detects dead processes and cleans up stranded resources.
- **RECEIVE** — Manages ECP worker processes.
- **ECPWork** — Performs ECP tasks (ECP worker process).
- **%SYS.SERVER** — Accepts TCP requests and dispatches workers to serve them (the superserver process).
- **%CSP.Daemon** — Manages expiration of web sessions.
- **LMFMON** — Monitors the InterSystems IRIS license and sends usage data to the license server over UDP.
- **%SYS.Monitor.xxx** — Performs system monitor tasks (various system monitor workers).
- **SYS.Monitor.xxx** — Writes alerts to alert file and transmits email alerts.

Do not stop the core system processes. Doing so disrupts the normal operation of an InterSystems IRIS instance.

A number of other InterSystems IRIS system processes are started after the core system processes. Many are started dynamically. These processes have a value displayed in the User column. Many of them are optional and are not started unless needed or configured. These processes can usually be identified by the values of the **Routine**, **User**, and **Client EXE** columns of the process display.

The task manager process (TASKMGR) is created during instance startup. It starts various scheduled system and user defined tasks and runs with the settings:

- Username = TASKMGR
- Routine = %SYS.TaskSuper.1
- Operating System Username = TASKMGR

If you are not using ECP, you can prevent the ECPWork process from being started:

1. From the Management Portal, select **System Administration** > **Configuration** > **Connectivity** > **ECP settings** and set the maximum number of application and data servers to zero.
2. Disable the ECP service.

ECP Server Processes

ECP server processes that are dynamically started have a routine name beginning with “ECP”. The user name or Operating System Username is usually **Daemon** or **%System**, but it may be the name of the Instance Service user on Windows. Examples of process names follow:

- ECPCliR – ECP client reader
- ECPCliW – ECP client writer
- ECPSrvR – ECP server reader
- ECPSrvW – ECP server writer

Web Server Processes

Web server processes are dynamically started. They will display CSPSystem in the **User** column when they are idle and waiting for a task. When they are active, they will display the InterSystems IRIS user for the web session and the current routine name. The **OS Username** column will display **Web Gateway**.

- %SYS.cspServer and %SYS.cspServer2 – Webserver routines that processes use to handle web application requests.
- %SYS.cspServer3 – Webserver routine that processes use to handle asynchronous communication and Web Gateway management.

These processes are associated with legacy applications from other InterSystems products. For more details on these routines in those applications, see the question about them in the [FAQ for this feature](#).

Note: These routines do not consume licenses. Licenses are associated with web application sessions.

For each of these servers, on Windows the executable is CSPAP.dll; on UNIX®, it is CSPap.so. The operating system username is **Web Gateway**. The program name may change as the process changes tasks.

Mirroring System Processes

Mirror system processes are started if mirroring is configured. They perform various functions related to mirroring.

- **MIRRORMGR** – Mirror Master. The User is the description of the mirror function performed: `Mirror Master`, `Mirror Primary`, `Mirror Dejournal`, `Mirror Prefetch`, or `Mirror JrnRead`. The operating system username is `Daemon`. No TCP port is open. The device is the operating system null device.
- **MIRRORCOMM** – Mirror communication process. The username is `Mirror Arbiter`, `Mirror Backup`, or `Mirror Svr:RdDmn`. The operating system username is `Daemon`. The device is `|TCP|XXX`. The TCP port can be determined from the Device name or the Mirror configuration.

IP Protocols

TCP

An InterSystems IRIS instance accepts connections on TCP/IP ports specified by configuration options. Any Operating System restrictions on usage of ports, for example with a fire wall, require port settings to allow inbound access that are consistent with the ports configured for InterSystems IRIS. If the firewall defines rules for executables, as it does on Windows, you may need to grant permission to programs as well, for example, the `irisdb.exe`, `licmanager.exe`, `ISCAgent.exe`, and the `httpd.exe` executables will require such permissions.

TCP/IP ports used by InterSystems IRIS are defined by the instance configuration. The configured ports can be examined in the `iris.cpf` file in the installation directory. The `[Startup]` section configures `DefaultPort`, `DefaultPortBindAddress` and `WebServerPort`. `DefaultPort` specifies the port on which the superserver accepts connections; the default value is 1972. `DefaultPortBindAddress` optionally specifies an interface address the superserver binds to. `WebServerPort` specifies the port on which the private web server accepts connections; the default value is 52773.

The private web server is mostly used in development environments and is not recommended for production environments.

The `[SQL]` section contains `JDBCGatewayPort` which defines the Java Database Connectivity (JDBC) gateway port number. Its default value is 62972.

The `[Telnet]` section contains a `Port` value to specify the port on which the InterSystems Telnet service (`ctelnetd.exe`) accepts Telnet connections to InterSystems IRIS on Windows.

UDP

InterSystems IRIS and the license server (`licmanager` or `licmanager.exe`) communicate primarily using the UDP protocol. InterSystems IRIS sends messages as UDP packets to the license server port. This port is 4002 by default, and is configured in the Management Portal > System Administration > Licensing > License Servers. The license server replies to InterSystems IRIS at the port that InterSystems IRIS used to send the original message (it looks up the port in the packet header). TCP is only used between InterSystems IRIS and the license server during a query request. InterSystems IRIS opens a TCP port for accept/listen and sends this port number in the query request. The license server connects back to that port and sends the results over the TCP connection. The port number is the license server port; if this fails, it uses port 0 which signals the operating system to select a free port at random. The port is open only during transmission of the query results.

SNMP

The `%System_Monitor` Service enables InterSystems IRIS to act as a subagent to an SNMP Agent on the managed system. This supports both SNMP requests (`GET` or `GETNEXT`) for InterSystems IRIS management and monitoring data (as defined in the supplied MIBs), and SNMP Traps (asynchronous notifications sent by InterSystems IRIS). Disabling the `%System_Monitor` service will disable all communication between InterSystems IRIS and the SNMP Agent on the local system, and consequently with any remote SNMP manager applications.

HTTP

Refer to the description of the components of the Web Gateway used by InterSystems IRIS to serve HTTP requests by navigating through the online documentation as follows: Documentation > InterSystems IRIS Web Development > Web Gateway Guide > Introduction to the Web Gateway. The private Web Server is `httpd.exe` (`httpd` on UNIX®) located in the

httpd\bin subdirectory under the installation directory. Startup of the private web server is controlled by the Management Portal > System Administration > Configuration > Additional Settings > Startup > WebServer set to true or false.

Gateways

InterSystems IRIS provides a number of Gateways to external data. These include SQL Gateway, JDBC Gateway, Object Gateways, and XSLT 2.0 Gateway servers. The TCP/IP ports used are defined using the gateway setup pages accessed via the Management Portal > System Administration > Configuration > Connectivity. See the documentation of these gateways for an explanation of Operating System services or processes on which they depend.

Remove Unneeded InterSystems IRIS Processes

InterSystems service processes are not created unless the services they support are enabled and configured. There is no need to take any additional action to prevent InterSystems service processes from running.

External Processes

An InterSystems IRIS instance will start processes running executables other than `iris[.exe]` to perform a number of functions in support of the instance. Instance specific versions of these executables, which are generally specific to the instance version, live in the bin subdirectory of the installation directory. Executables that may be shared by multiple InterSystems IRIS instances live in a common directory.

Persistent processes may be running the following executables, which live in the bin directory on Windows.

- `irisdb.exe` — The InterSystems IRIS executable.
- `licmanager.exe` — The InterSystems IRIS license server.
- `CStudio.exe` — Studio.
- `iristray.exe` — The InterSystems IRIS launcher in the system tray.
- `Iristerm.exe` — The Terminal.
- `iristrmd.exe` — The local Terminal connection daemon. Accepts local Terminal connections (not Telnet) and creates InterSystems IRIS server processes to serve the connection.
- `irisirdimj.exe` — Executable that processes the WIJ file during InterSystems IRIS startup and shutdown.

Persistent processes may be running the following executables, which live in the bin directory on UNIX®.

- `iris` — The InterSystems IRIS executable.
- `licmanager` — The InterSystems IRIS license server
- `irisirdimj` — Processes the WIJ file during InterSystems IRIS startup and shutdown.

Other programs in the bin directory are used from time to time, but the processes are short running and unlikely to be displayed by a process listing for long.

Executable binaries shared by InterSystems IRIS instances reside in subdirectories of `C:\Program Files (x86)\Common Files\InterSystems` on Windows. The processes may be seen running these executable binaries from the common directory on Windows.

- `ISCAgent.exe` – Controls mirror failover.
- `Iristerm.exe` – The Terminal.

Shared binaries are usually installed in `/usr/local/etc/irissys` on UNIX®.

- `ISCAgent*` - Controls mirror failover.

In addition to executable binaries, a number of shared library binaries are stored in the common directory.

Interoperability

Adapters

InterSystems IRIS provides communication with external interfaces using adapters.

Email

Email adapters are InterSystems IRIS processes. They use TCP/IP to send/receive email from an email server. Outbound adapters send mail to a SMTP server. Inbound adapters poll for relevant (filtered) messages from a POP3 server. Email servers are likely to be on a remote server, so while there would be no local process, the remote system would need to be reachable through a firewall.

File

File Input Adapters are InterSystems IRIS processes. They periodically inspect a directory they have been configured to monitor, read files that appear there, pass the files to the Business Service they have been configured to support, and move the files to the configured archive directory. The `EnsLib.File.InboundAdapter` class provides the implementation. The `FilePath`, `WorkPath`, and `ArchivePath` properties define the input, temporary work, and archive directories, respectively.

File Output Adapters are employed by production Business Operations to write data to files. The file path and file name are specified by the Business Operation and operations on the file are invoked by calling methods of the `EnsLib.File.OutboundAdapter` class. Messages are usually queued to a worker job that performs the actual output operation. This implies the existence of `Ens.Queue` processes.

FTP

InterSystems IRIS acts as a client for FTP communication with remote FTP servers using the `%Net.FtpSession` class. The `%Net.FtpSession` class can be configured to use PASV for the data channel to avoid an inbound connection. InterSystems IRIS provides FTP inbound and outbound adapters. Both act as FTP clients to get (input) or put (output) under the control of a Business Service created by the customer. The FTP server and port are configurable. The FTP adapters are InterSystems IRIS processes.

HTTP

The HTTP adapters (`EnsLib.HTTP.InboundAdapter` and `EnsLib.HTTP.OutboundAdapter`) enable productions to send and receive HTTP requests and responses. HTTP adapters are implemented by InterSystems IRIS processes. The port and interface IP addresses of the inbound HTTP adapter are configurable. The server and port to which the outbound HTTP adapter is provided by class settings.

Java Gateway

Production adapters use the Java Gateway to communicate through a Java intermediary process. A Java process is started which depends on the existence of a Java Virtual Machine. The InterSystems IRIS server process communicates with the Java process via a TCP connection. The TCP ports used are configurable.

LDAP

The `EnsLib.LDAP.OutboundAdapter` class can be used like other adapters by Business Services to send requests to an LDAP server and receive responses.

MQSeries

The classes `EnsLib.MQSeries.InboundAdapter` and `EnsLib.MQSeries.OutboundAdapter` enable productions to retrieve messages from and send messages to message queues of IBM WebSphere MQ. Dynamically loaded shared library binaries are used for the communication.

Pipe

The classes `EnsLib.Pipe.InboundAdapter` and `EnsLib.Pipe.OutboundAdapter` enable productions to invoke operating system commands or shell scripts. They create a process external to InterSystems IRIS and communicate with it via a pipe, so an external process will exist while the Pipe adapter is communicating with it. The command that the process runs is determined by the value assigned to the *CommandLine* property of the adapter class.

SAP

The Java Gateway is used to communicate with the SAP Java Connector using classes imported with the `EnsLib.SAP.Bootstrap` class `ImportSAP` method.

SQL

The SQL inbound and outbound adapters enable productions to communicate with JDBC or ODBC-compliant databases. In general, the inbound SQL adapter (`EnsLib.SQL.InboundAdapter`) periodically executes a query and then iterates through the rows of the result set, passing one row at a time to the associated business service. The SQL adapters use the underlying capabilities of InterSystems SQL and JDBC Gateways.

TCP

InterSystems IRIS provides input and output TCP adapters. Each TCP inbound adapter checks for data on a specified port, reads the input, and sends the input as a stream to the associated business service. Within a production, an outbound TCP adapter is associated with a business operation that you create and configure. The business operation receives a message from within the production, looks up the message type, and executes the appropriate method in the outbound TCP adapter to transmit the data over TCP.

Telnet

InterSystems IRIS provides the `EnsLib.Telnet.OutboundAdapter` which permits outbound telnet connections to the telnet facility on another system. This adapter provides methods to programmatically emulate the effect of manually logging in to the remote system using telnet client software. The InterSystems IRIS TCP device is the underlying technology.

Security Checklist

Checklist for Hardening Your Deployment

This checklist is intended to provide your organization with guidelines for assessing how secure your environment is and to provide tips for hardening your environment that will help your organization avoid and prevent security breaches. This checklist is not intended to be a “how to list” and is not all-inclusive. The points below are items to consider rather than a definitive list of rules to apply.

You alone are responsible for the security of your infrastructure. If you are uncertain about your approach to hardening and protection, consult a security professional.

Network and Firewalls

ID	Topic	Description
1.	Network, hardware, software and policies	Obtain copies of and review security policies, firewall logs, firewall configuration and patch levels, public facing IP addresses, diagrams of network, and firewall topologies.
2.	Auditing the physical environment	Ensure firewalls and management servers are in a physically secure location that can only be accessed by authorized personnel. Ensure that they are patched up to date.
3.	Reviewing change management process, rule base modifications	Review procedures and approval process for changes. Automation tools are available for this.
4.	Vulnerability testing	Run automated tools to analyze and identify unsecured services, protocols, and ports.
5.	Using brute force detection systems	Stop people from guessing passwords, and prevent them from connecting to the server, by blocking their current IP address in your server firewall.
6.	Ongoing audits and real-time monitoring and alerting	Ensure a process is in place for continuous auditing of firewalls. Ensure real-time monitoring is in place to alert on changes to the firewall. Review their logs regularly.

Operating System

ID	Topic	Description
1.	Installation planning	Understand the server role, and document the install procedure. Download appropriate operating system securing and hardening guides for more detailed information.
2.	Patch levels	Ensure operating system patches are up to date, especially security patches. Turn off automatic updates.
3.	Endpoint protection software	Install and appropriately configure this software. (Formerly listed as antivirus software.)
4.	Disabling unnecessary software, services, and ports	<p>Disable unnecessary network services such as IPv6, telnet, and FTP.</p> <p>Disable unnecessary daemons that are not used such as DHCP, scheduling and queuing services, and laptop services.</p> <p>Configure in-use services to be as secure as possible; for example, secure SSH by limiting SSH protocol to Version 2 (Version 1 is not secure).</p>
5.	Logs	Maintain server logs and mirror those logs to a separate log server.
6.	Monitoring and alerting	Configure monitoring and alerting settings to notify of events such as changes to the system, and unauthorized access.
7.	Physical security	Configure the BIOS to disable booting from CDs/DVDs, floppies, and external devices; set a password to protect these settings.

Web Server

ID	Topic	Description
1.	Installation planning	Understand the role of the web server: what content will it serve; will the pages be static; what web services are provided? Document the installation procedure. Download and review the appropriate hardening security guide.
2.	Patch levels	Ensure web server is up to date, especially with regard to security patches.
3.	Web server header info	Configure the servers so that HTTP headers do not provide information relating to the web server software being run, or system types and versions.
4.	Disabling HTTP TRACE	When enabled, HTTP TRACE request is used to echo back all received information.
5.	Error handling	Implement proper error handling by utilizing generic error pages and error handling logic to force the application to avoid default error pages. These often leak sensitive system and application information.
6.	Disabling modules	Disable all unused modules to reduce surface area of the web server; these modules often provide too much information – <i>Apache</i> : autoindex, cgi, imap, info, status, userdir, actions, negotiation... <i>IIS</i> : ASP, ASP.NET, WebDAV, CGI, directory browsing...
7.	Users and groups	<i>Apache</i> : Run Apache as a separate user and group so Apache processes cannot be used by other system processes. <i>IIS</i> : Remove unused accounts; disable Guest account

Users, Passwords, Groups, Ownerships, and Permissions

ID	Topic	Description
1.	User management	Disable root login. All administrators should be named users. Regularly check for unused user accounts, and for default user accounts and passwords.
2.	Password policy	Require and use very strong passwords with mixed case, numbers, and special characters. Change passwords on a regular basis. Lock accounts after too many login failures.
3.	UNIX®	Create groups and users before installation. Install InterSystems IRIS as root. Ensure groups, ownerships, and permissions for InterSystems IRIS databases are maintained as specified.
4.	Windows	Install InterSystems IRIS using the Windows Administrator, and then disable the default Windows Administrator account. Also disable Guest and Help Assistant accounts.

Encryption (Data At Rest and Data In Motion)

ID	Topic	Description
1.	Data at rest	Ensure all production data at rest on disk is encrypted.
2.	Key management	Review the key management policies and procedures.
3.	Data In motion	Ensure all HTTP data communications is encrypted, such as with TLS. Ensure that all TLS configurations are using the latest version.

InterSystems Security

ID	Topic	Description
1.	Installation	Always install with the Locked Down initial security setting type.
2.	Authentication	Regularly review users and passwords.
3.	Authorization	Review application requirements; define roles, resources, and services.
4.	Auditing	Ensure that auditing is enabled. Review the logs regularly.
5.	Disabling services	If services such as ECP and mirroring are not used, do not enable them.
6.	Removing unused databases and applications.	Remove unused databases such as USER.

Identity and Access Management

Identity and Access Management Introduction

Introduction to InterSystems Identity and Access Management

Identity and access management (IAM) is a framework focused on managing user identities and access permissions on a computer network. This includes *authentication* and *authorization*. Authentication is the process of verifying the identity of a user attempting to connect to InterSystems IRIS. InterSystems IRIS supports several different authentication mechanisms. Once authenticated, a user can communicate with InterSystems IRIS and use its tools and resources. Authorization is the process of determining which database assets a user can use, view, or change.

Kerberos Authentication

About Kerberos Authentication

Kerberos Background

For maximally secure connections, InterSystems IRIS supports the *Kerberos* authentication system, which provides a highly secure and effective means of verifying user identities. Kerberos was developed at the Massachusetts Institute of Technology (MIT) to provide authentication over an unsecured network, and protects communications using it against sophisticated attacks. The most evident aspect of this protection is that a user's password is never transmitted over the network — even encrypted.

Kerberos is what is called a *trusted-third-party system*: the Kerberos server holds all sensitive authentication information (such as passwords) and is itself kept in a physically secure location.

Kerberos is also:

- Time-tested — Kerberos was originally developed in the late nineteen-eighties. Its principal architecture and design have been used for many years at many sites; subsequent revisions have addressed issues that have been discovered over the years.
- Available on all supported InterSystems IRIS platforms — Originally developed for UNIX®, Kerberos is available on all InterSystems IRIS-supported variants of UNIX®; Microsoft has integrated Kerberos into Windows 2000 and subsequent versions of Windows. (Note that because the Microsoft .NET framework does not include direct Kerberos support, InterSystems IRIS does not support Kerberos for the InterSystems IRIS Managed Provider for .NET.)
- Flexibly configurable — It accommodates heterogeneous networks.
- Scalable — The Kerberos protocol minimizes the number of interactions with its Key Distribution Center (KDC); this prevents such interactions from becoming a bottleneck on larger systems.
- Fast — As an open-source product, the Kerberos code has been scrutinized and optimized extensively over the years.

Underlying Kerberos authentication is the AES encryption algorithm. AES — the Advanced Encryption Standard — is a royalty-free, publicly defined symmetric block cipher that supports key sizes of 128, 192, and 256 bits. It is part of the US Federal Information Processing Standard (FIPS), as chosen by United States National Institute of Standards and Technology (NIST).

For background on Kerberos, see the [MIT Kerberos website](#) and [its list of available documentation](#).

How Kerberos Works

In the Kerberos model, there are several different actors. All the different programs and people being authenticated by Kerberos are known as *principals*. The Kerberos system is administered by a Kerberos Key Distribution Center (KDC); on Windows, the Windows Domain Controller performs the tasks of a KDC. The KDC issues tickets to users so that they can interact with programs, which are themselves represented by *service principals*. Once a user has authenticated and has a service ticket, it can then use a program.

Specifically, Kerberos authentication involves three separate transactions:

1. The client receives what is called a “ticket-granting ticket” (“TGT”) and an encrypted session key.
2. The client uses the TGT and session key to obtain both a service ticket for InterSystems IRIS as well as another encrypted session key.
3. The client uses the service ticket and second session key to authenticate to InterSystems IRIS and optionally establish a protected connection.

Aside from a possible initial password prompt, this is designed to be invisible to the user.

How InterSystems IRIS Uses Kerberos

To ensure that Kerberos properly secures an environment, all InterSystems services that support it must have Kerberos enabled and those that don't support it must be disabled. The exception to this is that services intended to operate within the InterSystems security perimeter, such as ECP, do not support Kerberos; you can simply enable or disable these services, since they are designed for use in an externally secured environment.

Overview of Configuring Kerberos

To configure an InterSystems IRIS instance for Kerberos authentication:

1. Ensure that InterSystems IRIS is set up to run as a Kerberos service.

The procedure varies, depending on the operating system of the InterSystems IRIS server and the type of environment; see [Preparing the Security Environment for Kerberos](#) for more information.
2. Enable the relevant Kerberos mechanisms on the **Authentication/Web Session Options** page (**System Administration** > **Security** > **System Security** > **Authentication/Web Session Options**).
3. Determine which services will be used to connect to InterSystems IRIS and disable all other services. For a list of which services are used by what connection tools, see the table [Connection Tools, Their Access Modes, and Their Services](#).
4. For client-server connections, specify what Kerberos connection security level the server requires. This is how you determine which Kerberos features are to be part of connections that use the service. See [Specify Connection Security Levels](#) for more information.
5. For client-server connections, perform client-side setup. This ensures that the application has access to the information it needs at runtime. This information includes:
 - The name of the service principal representing InterSystems IRIS.
 - The allowed connection security levels.

Setting up this information may involve configuring a Windows preferred server or some other configuration mechanism. See [Set Up a Client](#) for more information.

6. Specify how the authentication process obtains user credentials. This is either by checking the user's Kerberos credentials cache or by providing a Kerberos password prompt for the user. See [Obtain User Credentials](#) for more information.
7. To maximally secure web connections, set up [secure channels](#) for the each connection.

Important: On Windows, when logged in using a domain account, OS-based and Kerberos authentication are the same. When logged on locally, Kerberos is subject to a KDC spoofing attack and is therefore neither secure nor recommended.

About Kerberos and the Access Modes

Each connection tool uses a service to establish communications with InterSystems IRIS. It also uses a particular access mode. To ensure maximum protection, determine which services you need, based on which connection tools you are using. If you are not using a service, disable it.

The following is a list of connection tools, their access modes, and their services:

Table B-1: Connection Tools, Their Access Modes, and Their Services

Connection Tool	Access Mode	Service
InterSystems IRIS Telnet	Client-Server	%Service_Telnet
CallIn	Local	%Service_CallIn
Console	Local	%Service_Console
Java	Client-Server	%Service_Bindings
JDBC	Client-Server	%Service_Bindings
ODBC	Client-Server	%Service_Bindings
Terminal	Local	%Service_Terminal
Web technologies	Web	%Service_WebGateway

Local

Kerberos authentication for a local service establishes that the user and InterSystems IRIS are both valid Kerberos principals. There is only one machine in use and only one process on that machine; hence, the configuration pages for these services in the Portal allow you to specify whether to use Kerberos prompting (labeled simply as Kerberos in the Management Portal) or Kerberos credentials cache.

In this scenario, there is no *connection* between the user and InterSystems IRIS, since both are using the same process on the same machine. Because the two are sharing a process, there is no information being passed through an unsecured medium and therefore no need to offer special protections for this data. (This situation is known as *in-process authentication*.)

Client-Server

Client-server applications include connections from Java, JDBC, ODBC, and through Telnet. For a client-server application using Kerberos authentication, the user needs credentials to interact with InterSystems IRIS via the application.

The server and client each require configuration. Server configuration specifies which type of connections are accepted; client configuration specifies what type of connection is attempted and may also specify how to obtain the user's credentials.

With client-server connections, Kerberos supports various connection security levels, which are configured on the InterSystems IRIS server machine:

- **Kerberos** — Kerberos manages the initial authentication between the user and InterSystems IRIS. Subsequent communications are not protected.
- **Kerberos with Packet Integrity** — Kerberos manages the initial authentication between the user and InterSystems IRIS; each subsequent message has a hash that provides source and content validation. This provides verification that each message in each direction is actually from its purported sender; it also provides verification that the message has not been altered in transit from sender to receiver.
- **Kerberos with Encryption** — Kerberos manages initial authentication, ensures the integrity of all communications, and also encrypts all communications. This involves end-to-end encryption for all messages in each direction between the user and InterSystems IRIS.

Web

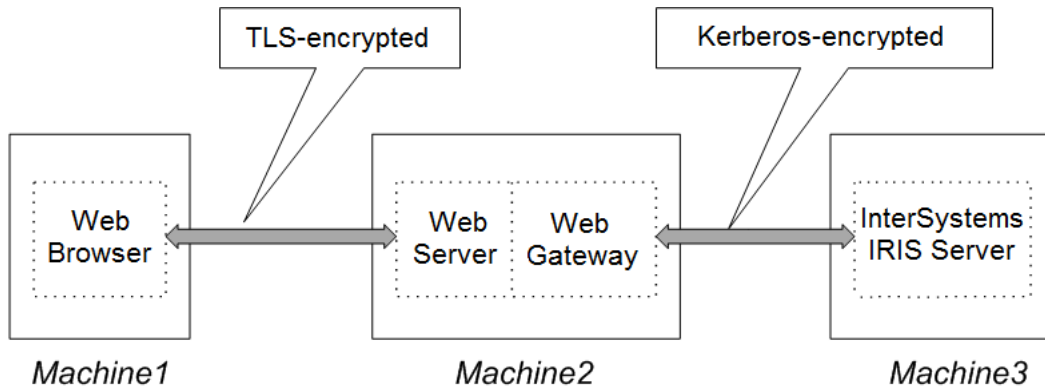
When running a web application, the user does not interact directly with the InterSystems IRIS server. To protect all information from monitoring, you need to encrypt the connections between the user and InterSystems IRIS as follows:

- Configure the web server so that it uses TLS to secure browser connections to it.

- Co-locate the web server and the web gateway, so there is no need to secure the connection between them. If this is not possible, then ensure the data on the connection is secure through another way.
- Configure the web gateway to use Kerberos authentication and encryption. Use the gateway's Kerberos principal to establish such a connection.

The architecture is:

Figure B–1: Architecture of a Kerberos-Protected Web Connection



Any communications between the end-user and InterSystems IRIS occurs through TLS-encrypted or Kerberos-encrypted pipes. For Kerberos-secured connections, this includes the end-user's Kerberos authentication.

Because the InterSystems IRIS server cannot prompt the end-user for a password, it invokes an API that sends HTML content to the browser to prompt. The user completes this form that has been sent; it travels back to the web server, which hands it to the web gateway, which then hands it to the InterSystems IRIS server. The web server acts as a proxy on behalf of the user at the browser; this is why this kind of a connection is known as a *proxy* connection. At the same time, all information related to the user resides on the server machine (as with the local access mode); hence a web connection is also a form of in-process authentication.

Specify Connection Security Levels

Client-server connections to InterSystems IRIS use one of the following services:

- **%Service_Bindings** — Java, JDBC, ODBC
- **%Service_Telnet** — Telnet

For any Kerberos connection using one of these services, you must specify the connection security levels which the server accepts. To configure the service's supported connection security levels, the procedure is:

1. On the **Authentication/Web Session Options** page (**System Administration > Security > System Security > Authentication/Web Session Options**), specify which connection security levels to enable for the entire InterSystems IRIS instance, where these can be:
 - **Kerberos** — Initial authentication only
 - **Kerberos with Packet Integrity** — Initial authentication and packet integrity
 - **Kerberos with Encryption** — Initial authentication, packet integrity, and encrypting all messages

For more information on the **Authentication Options** page, see [Authentication Options](#).

2. On the **Services** page (**System Administration > Security > Services**), click the service name (in the **Name** column); this displays the **Edit Service** page for the service.

3. On the **Edit Service** page, specify which connection security levels to require as part of a Kerberos connection. After making this selection, click **Save**.

If a client attempts to connect to the server using a lower level of security than that which is specified for the server, then the connection is not accepted. If a client attempts to connect to the server using a higher level of security than that which is specified for the server, then the server connection attempts to perform authentication using the level of security that it specified.

Set Up a Client

When using the client-server access mode, you need to configure the client. The particulars of this process depend on the connection technology being used.

Telnet: Set Up the Preferred Server for Use with Kerberos

With a Windows client, when establishing a connection using InterSystems IRIS telnet for Windows, the client uses configuration information that has been stored as part of a remote server.

Important: InterSystems IRIS has its own telnet server for Windows. When connecting to a non-Windows machine, there is no InterSystems IRIS telnet server available — you simply use the telnet server that comes with the operating system. Once you have established the connection to the server machine, you can then start InterSystems IRIS using the **%Service_Terminal** service.

To configure a client connection coming in through telnet go to the client machine. On that machine, the procedure is:

1. Click on the InterSystems IRIS launcher and select **Preferred Server** from the menu (the **Preferred Server** choice also displays the name of the current preferred server).
2. From the submenu that appears, choose **Add/Edit**.
3. To create a new remote server, click the **Add** button; to configure an already-existing server, choose the InterSystems IRIS server to which you are connecting and click the **Edit** button.
4. This displays the **Add Connection** dialog. In the **Authentication Method** area on that dialog, click **Kerberos**. This expands the dialog to display a number of additional fields.
5. If you are editing the values for an already-existing server, there should be no need to change or add values for the more general fields in this dialog, as they are determined by the server that you chose to edit.

If you are adding a new server, the fields to complete are described in [Define a Remote Server Connection](#).

6. In the dialog's Kerberos-related fields, specify values for the following fields:
 - The connection security level, where the choices are Kerberos authentication only; Kerberos authentication with packet integrity; or Kerberos authentication, packet integrity, and encryption
 - The service principal name. For information on setting up service principal names, see [Names and Naming Conventions](#).
 - If you are configuring a telnet connection to a Windows machine, check the box specifying that the connection use the Windows InterSystems IRIS Telnet server.
7. Click **OK** to save the specified values and dismiss the dialog.

Set Up an ODBC DSN with Kerberos

InterSystems IRIS supports Kerberized ODBC connections from clients on Windows, UNIX®, and Mac to DSNs (Data Source Nodes) on all platforms. The ways of configuring client behavior vary by platform:

- On all platforms, the **SQLDriverConnect** function is available, which accepts a set of name-value pairs. **SQLDriverConnect** is a C call that is part of the ODBC API and is documented at the [Microsoft website](#). Its name-value pairs are the same as those for the initialization file available on non-Windows platforms.
- On non-Windows platforms, use the InterSystems ODBC initialization file to specify name-value pairs that provide connection information. This file is described generally in Using the InterSystems ODBC Driver. The file has the following Kerberos-related variables:
 - *Authentication Method* — Specifies how the ODBC client authenticates to the DSN. 0 specifies instance authentication; 1 specifies Kerberos.
 - *Security Level* — For Kerberos connections, specifies which functionality is used to protect the connection. 1 specifies that Kerberos is used for authentication only; 2 specifies that Kerberos is used for authentication and to ensure the integrity of all packets passed between client and server; and 3 specifies that Kerberos is used for authentication, packet integrity, and to encrypt all messages.
 - *Service Principal Name* — Specifies the name of InterSystems service that is serving as the DSN. For example, the service principal might have “iris/localhost.domain.com” as its name.

The names of these variables must have spaces between the words. They are not case-sensitive.

- On a Windows client, you can specify connection information through a GUI: the ODBC DSN configuration dialog. InterSystems IRIS provides options on the **System DSN** tab. This screen has associated help that describes its fields. The path on the Windows Start menu to display this screen varies by version of Windows; it may be listed under **Administrative Tools**.

Important: On 64-bit Windows, there are two versions of `odbcad32.exe`: one is located in the `C:\Windows\System32\` directory and the other is located in the `C:\Windows\SysWOW64\` directory. If you are running 64-bit Windows, configure DSNs through the one in `C:\Windows\SysWOW64\`.

Set Up a Java or JDBC Client with Kerberos

InterSystems IRIS provides a Java class that serves as a utility to assist with Java client configuration. Run it when you are ready to configure the client. The procedure is:

1. To configure the client for use with Kerberos, issue the Java **Configure** command such as:

```
java -classpath '$IRIS_INSTALL_DIRECTORY/dev/java/lib/JDK18/*' com.intersystems.jgss.Configure
```

This allows **Configure** to run from any location on the machine, not just from within the JDK directory. Note that the specifics of this command may vary, depending on your site, such as to accommodate Windows path styles or using JDK11.

This program uses Java Generic Security Services (JGSS) to perform the following actions:

- If necessary, modifies the `java.security` file.
 - Creates or modifies the `isclgin.conf` file.
2. The program then prompts you to create and configure the `krb5.conf` file. If the file exists, the command prompts if you wish to use the existing `krb5.conf` or replace it; if you choose to replace it, it prompts for the following information:
 - a. Kerberos realm — It offers the local domain in lowercase as a default value for the domain.
 - b. Primary KDC — You only need include the local machine name, as the program appends the Kerberos realm name to the machine name for you.
 - c. Secondary KDC(s) — You can specify the names of zero or more KDCs to replicate the content of the primary KDC.

3. After receiving this information, run the command a second time. (It instructs you to do this.)
4. When prompted to replace `krb5.conf`, choose to leave the existing file. The command then tests the connection by prompting for the username and password of a principal in the specified Kerberos realm.

If this succeeds, then client configuration is complete.

Obtain User Credentials

For all access modes, you need to specify whether the application obtains the user's credentials from an existing credentials cache or by prompting for a username and password.

Obtain Credentials for Local Access Mode

For the local access mode, the user's credentials reside on the same machine as InterSystems IRIS. In this situation, the application is using a service to connect to InterSystems IRIS. This includes the following services:

- `%Service_CallIn`
- `%Service_Console`
- `%Service_Terminal`

To specify how to get credentials, the procedure is:

1. On the **Services** page (**System Administration** > **Security** > **Services**) and select the service from the **Name** column. This displays the **Edit Service** page for the service.
2. On the **Edit Service** page, specify how to get credentials. Either select prompting (the **Kerberos** check box) or by using a credentials cache (the **Kerberos Credentials Cache** check box). Do not mark both.

Click **Save** to use the settings.

Note: If you enable both Kerberos (prompting) and Kerberos credentials cache authentication for the service, then the credentials cache authentication takes precedence. This is behavior specified by Kerberos, not InterSystems IRIS.

On Windows with a Domain Controller (the likely configuration for Windows), logging in establishes a Kerberos credentials cache. On UNIX®, Linux, and macOS, the typical default condition is to have no Kerberos credentials, so that InterSystems IRIS is then configured to use Kerberos prompting; on these systems, the user can obtain credentials in either of the following ways:

- Running **kinit** before invoking the Terminal
- Logging in to a system where the login process performs Kerberos authentication for the user

In these situations, InterSystems IRIS can be configured to use the credentials cache.

Obtain Credentials for Client-Server Access Mode

For client-server access mode, the user's credentials reside on the machine that hosts the client application. In this case, the manner in which you specify how to obtain credentials varies according to how the client is connecting:

- ODBC and Telnet
- Java and JDBC

ODBC and Telnet

The underlying InterSystems IRIS code used by these connection tools assumes that end-users already have their credentials; no prompting is necessary.

On Windows, every user logged on in the domain has a credentials cache.

On other operating systems, a user has a credentials cache if the operating system has performed Kerberos authentication for the user, or if the user has explicitly run **kinit**. Otherwise, the user has no credentials in the cache and the connection tool fails authentication.

Note: Not all connection tools are available on all operating systems.

Java and JDBC

When using Java and JDBC, there are two different implementations of Java available — either Oracle or IBM. These have several common behaviors and several differing behaviors.

Note: IBM implementations of Java are available through version 8 only; for later versions, [IBM supports open source versions](#).

Both implementations store information about a connection in properties of an instance of the `java.util.Properties` class. These properties are:

- *user* — The name of the user who is connecting to the InterSystems IRIS server. This value is only set for certain connection behaviors.
- *password* — That user's password. This value is only set for certain connection behaviors.
- *service principal name* — The Kerberos principal name for the InterSystems IRIS server. This value is set for all connection behaviors.
- *connection security level* — The type of protection that Kerberos provides for this connection. 1 specifies that Kerberos is used for authentication only; 2 specifies that Kerberos is used for authentication and to ensure the integrity of all packets passed between client and server; and 3 specifies that Kerberos is used for authentication, packet integrity, and to encrypt all messages. This value is set for all connection behaviors.

In the following discussions, the instance of the `java.util.Properties` class is referred to as the *connection_properties* object, where the value of each of its properties is set with a call to the **connection_properties.put** method, such as

```
String principalName = "MyServer";
connection_properties.put("service principal name",principalName);
```

For both implementations, credentials-related behavior is determined by the value of a parameter in the `isclogin.conf` file (see [Set Up a Java or JDBC Client for Use with Kerberos](#) for more information on this file).

There are two differences between the behavior of the two Java implementations:

- To specify credentials-related behavior, the parameter name to set in the `isclogin.conf` file differs for each implementation:
 - For IBM, it is *useDefaultCcache*.
 - For Oracle, it is *useTicketCache*.
- There are different behaviors available on each implementation. These are described in the following sections.

Specifying Behavior on a Client Using the IBM Implementation

The options are:

- To use a credentials cache, set the value of the *useDefaultCcache* parameter to `TRUE` and do not set the values of the *user* or *password* properties. Note that if no credentials cache is available, then an exception is thrown.
- To use a username and password that are passed in programmatically, set the value of the *useDefaultCcache* parameter to `FALSE` and set the values of the *user* and *password* properties.

- To prompt for a username and password, set the value of the *useDefaultCcache* parameter to FALSE and do not set the values of the *user* or *password* properties. Because these properties do not have values set, classes from libraries supplied with InterSystems IRIS can be used to generate prompts for them.

Specifying Behavior on a Client Using the Oracle Implementation

The options are:

- To exclusively use a username and password that are passed in programmatically, set the value of the *useTicketCache* parameter to FALSE and set the values of the *user* and *password* properties.
- To exclusively prompt for a username and password, set the value of the *useTicketCache* parameter to FALSE and do not set the values of the *user* or *password* properties. Because these properties do not have values set, classes from libraries supplied with InterSystems IRIS can be used to generate prompts for them.
- To exclusively use a credentials cache, set the value of the *useTicketCache* parameter to TRUE. To prevent any further action, set the values of the *user* and *password* properties to bogus values; this prevents prompting from occurring and ensures the failure of any authentication attempt based on the properties' values.
- To attempt to use a credentials cache and then fall through to using a username and password that are passed in programmatically, set the value of the *useTicketCache* parameter to TRUE and set the values of the *user* and *password* properties. If there is no credentials cache, then the properties' values are used.
- To attempt to use a credentials cache and then fall through to prompting for a username and password, set the value of the *useTicketCache* parameter to TRUE and do not set the values of the *user* or *password* properties. If there is no credentials cache, then classes from libraries supplied with InterSystems IRIS can be used to generate prompts for them.

Obtain Credentials for Web Access Mode

With a web-based connection that uses Kerberos, there is always a username and password prompt. If these result in authentication, the user's credentials are placed in memory and then discarded when no longer needed.

Set Up a Secure Channel for a Web Connection

To maximally secure a web connection, it is recommended that the two legs of communication — both between the browser and the web server and then between the web gateway and InterSystems IRIS — use secure channels. This ensures that any information, such as Kerberos usernames and passwords, be protected in transmission from one point to another. To secure each communications channel, the procedure is:

- [Between the web browser and web server](#)
- [Between the Web Gateway and InterSystems IRIS](#)

Secure the Connection Between a Web Browser and Web Server

The typical means of securing a connection between a web browser and a web server is to use TLS (Transport Layer Security).

Set Up a Kerberized Connection from the Web Gateway to InterSystems IRIS

To set up a secure, encrypted channel between the web gateway and the InterSystems IRIS server, you need a Kerberos principal that represents the gateway. This principal establishes an encrypted connection to InterSystems IRIS, and all information is transmitted through the connection. This allows an end-user to authenticate to InterSystems IRIS and prevents any snooping during that process.

Note: For information on setting up a connection between the web gateway and the InterSystems IRIS server that is protected by TLS, see [Configuring the Web Gateway to Connect to InterSystems IRIS Using TLS](#).

The procedure is:

1. Determine or choose the name of the Kerberos principal that represents the gateway.

For Windows, this is the principal name representing the gateway host's network service session (that is, the name of the machine hosting the gateway with the "\$" appended to it — *machine_name\$*, , such as Athens\$). For other platforms, this is any valid principal name entered as the username in the gateway configuration screen; this identifies the appropriate key in the key table file.

2. Create a user in InterSystems IRIS with the same name as the gateway's Kerberos principal. To do this, follow the instructions in [Create a New User](#).
3. Give that user permissions to use, read, or write any required resources (these are also known as privileges). This is done by [associating those privileges with a role](#) and then [associating the user with the role](#).
4. Configure the `%Service_WebGateway` service. To do this, complete the fields described in [Service Properties](#).
5. Configure the gateway so that it can contact the server. The procedure is:
 - a. From the Management Portal home page, go to the **Web Gateway Management** page (**System Administration > Configuration > Web Gateway Management**).
 - b. On the web gateway management page, there are a set of choices on the left. Under **Configuration**, click **Server Access**. This displays the **Server Access** page.
 - c. On the **Server Access** page, you can add a new configuration or edit an existing one. To add a new configuration, click the **Add Server** button; to edit an existing one, select it from the list on the left, select the **Edit Server** radio button, and click **Submit**. This displays the page for editing or configuring server access parameters. In addition to the general parameters on this page (described on its help screen), this page allows you to specify security-related parameters for the gateway. For Kerberos connections, these are:
 - **Connection Security Level** — Choose the kind of protection that you would like Kerberos to attempt to provide this connection. (Note that this must match or exceed the type of security specified for the web service in the previous step.)
 - **User Name** — The name of the Kerberos principal that represents the gateway. (This must be the same principal as was used in the first step of this process.)
 - **Password** — Do not specify a value for this. (This field is used when configuring the gateway for use with instance authentication.)
 - **Product** — InterSystems IRIS.
 - **Service Principal Name** — The name of the principal that represents the InterSystems IRIS server. This is typically a standard Kerberos principal name, of the form "iris/machine.domain", where *iris* is a fixed string indicating that the service is for InterSystems IRIS, *machine* is the machine name, and *domain* is the domain name, such as "intersystems.com".
 - **Key Table** — When connecting to an instance of InterSystems IRIS on Windows, leave this field blank; for other operating systems, provide the name of the keytab file containing the permanent key belonging to the web gateway, including the full path.

After entering all these values, click the **Save Configuration** button to save them.

The web service is now ready to configured. This means that it can now provide the necessary underlying infrastructure to support a web application.

When creating a secured web application, the application developer needs to:

1. Choose an authentication method.
2. Configure the roles for the application.

3. Make sure the browser-to-web server connection uses TLS.

Operating System–Based Authentication

About OS-Based Authentication

InterSystems IRIS supports what is called *operating system–based* (or *OS-based*) authentication. With operating system authentication, InterSystems IRIS uses the operating system’s user identity to identify the user for InterSystems IRIS. When operating system authentication is enabled, the user authenticates to the operating system according to the operating system’s protocols. For example, on UNIX®, this is traditionally a login prompt where the operating system compares a hash of the password to the value stored in the `/etc/passwd` file. When the user first attempts to connect to InterSystems IRIS, InterSystems IRIS acquires the process’ operating system level user identity. If this identity matches an InterSystems IRIS username, then that user is authenticated.

This capability only applies to server-side processes, such as terminal-based applications (for example, connecting through the Terminal) or batch processes started from the operating system. It is not available for an application that is connecting to InterSystems IRIS from another machine, such as when a copy of Studio on one machine is connecting to an InterSystems IRIS server on another.

This mechanism is typically used for UNIX® systems, in addition to the Windows console.

OS-based authentication is only available for local processes, namely:

- Callin (`%Service_Callin`)
- Console (`%Service_Console`)
- Terminal (`%Service_Terminal`)

Configuring OS-Based Authentication

To set up the use of this type of authentication, the procedure is:

1. On the **Authentication/Web Session Options** page (**System Administration** > **Security** > **System Security** > **Authentication/Web Session Options**), select **Allow Operating System authentication**.
2. On to the **Services** page (**System Administration** > **Security** > **Services**) and select the service from the **Name** column. This displays the **Edit Service** page for the service.
3. On the **Edit Service** page, choose operating system–based (the **Operating System** check box).
Click **Save** to use the settings.

This type of authentication requires no other configuration actions.

Note: On Windows, when logged in using a domain account, OS-based and Kerberos authentication are the same.

A Note on %Service_Console

Since the console (`%Service_Console`) is a Windows-based service and Windows domain logins typically use Kerberos, console’s OS-based authentication provides authentication for local logins.

A Note on %Service_Callin

With callin (`%Service_Callin`), OS-based authentication is only available from an OS-level prompt. When using callin programmatically, OS-based authentication is not supported — only unauthenticated access is available.

Instance Authentication

About Instance Authentication

InterSystems IRIS itself can provide a login mechanism, called *instance authentication*. (In the Management Portal, it is referred to as **Password Authorization**.) Specifically, InterSystems IRIS maintains a password value for each user account and compares that value to the one provided by the user at each login. As with traditional OS-based authentication, InterSystems IRIS stores a hashed version of the password. When the user logs in, the password value entered is hashed and the two hashed versions are compared. The system manager can configure certain password criteria, such as minimum length, to ensure a desired degree of robustness in the passwords selected by users. The criteria are described in [Password Strength and Password Policies](#).

InterSystems IRIS stores only irreversible cryptographic hashes of passwords. The hashes are calculated using the PBKDF2 algorithm with the HMAC-SHA-512 pseudorandom function, as defined in Public Key Cryptography Standard #5 v2.1: “Password-Based Cryptography Standard.” The current implementation uses 10,000 iterations, 64 bits of salt, and generates 64-byte hash values; to specify a different algorithm or increase the number of iterations, use the **Security.System.PasswordHashAlgorithm** and **Security.System.PasswordHashWorkFactor** methods, respectively. There are no known techniques for recovering original passwords from these hash values.

The services available for authentication with instance authentication are:

- **%Service_Binding**
- **%Service_CallIn**
- **%Service_ComPort**
- **%Service_Console**
- **%Service_Telnet**
- **%Service_Terminal**
- **%Service_WebGateway**

Overview of Configuring Instance Authentication

For a service to use instance authentication, you must configure it as follows:

1. On the **Authentication/Web Sessions Options** page (**System Administration > Security > System Security > Authentication/Web Session Options**), enable authentication with instance authentication by selecting **Allow Password authentication**.
2. For the particular service, go to the **Services** page (**System Administration > Security > Services**) and select that service, such as **%Service_Bindings**, in the **Name** column; this displays the **Edit Service** page for the service.
3. On this page, choose instance authentication, listed simply as **Password** from the list of authentication types.
4. Click **Save** to save this setting.
5. In addition to this basic procedure, certain services require further configuration. This is described in the following sections:
 - [Web](#)
 - [ODBC](#)
 - [Telnet](#)

Web

For web access, you can optionally require that the web gateway authenticate itself to the InterSystems IRIS server through instance authentication. To perform this configuration, the procedure is:

1. From the Management Portal home page, go to the **Web Gateway Management** page (**System Administration > Configuration > Web Gateway Management**).
2. On the web gateway management page, there are a set of choices on the left. Under **Configuration**, click **Server Access**. This displays the **Server Access** page.
3. On the **Server Access** page, you can add a new configuration or edit an existing one. To add a new configuration, click the **Add Server** button; to edit an existing one, select it from the list on the left, select the **Edit Server** radio button, and click **Submit**. This displays the page for editing or configuring server access parameters. In addition to the general parameters on this page (described on its help screen), this page allows you to specify security-related parameters for the gateway. For instance authentication connections, these are:
 - **Connection Security Level** — Choose **Password** from the drop-down list to use instance authentication.
 - **User Name** — The user name under which the gateway service runs (the installation process creates the CSPSystem user for this purpose). This user (CSPSystem or any other) should have no expiration date; that is, its Expiration Date property should have a value of 0.
 - **Password** — The password associated with the user account just entered.
 - **Product** — InterSystems IRIS.
 - **Service Principal Name** — Do not specify a value for this. (This field is used when configuring the gateway for use with Kerberos.)
 - **Key Table** — Do not specify a value for this. (This field is used when configuring the gateway for use with Kerberos.)

After entering all these values, click the **Save Configuration** button to save them.

It is important to remember that the authentication requirements for the gateway are not directly related to those for an application that uses the gateway. For example, you can require instance authentication as the authentication mechanism for a web application, while configuring the gateway to use Kerberos authentication — or no authentication at all. In fact, choosing a particular authentication mechanism for the gateway itself makes no technical requirement for the web application, and vice versa. At the same time, some pairings are more likely to occur than others. If a web application uses Kerberos authentication, then using any other form of authentication for the gateway means that Kerberos authentication information will be flowing through an unencrypted channel, thereby potentially reducing its effectiveness.

With a web application that uses instance authentication, the username and password of the end-user are passed from the browser to the web server, which then hands them to the web gateway. Since the gateway has its own connection to the InterSystems IRIS server, it then passes the username and password to the InterSystems IRIS server. To establish its connection to the InterSystems IRIS server, the gateway uses the CSPSystem account, which is one of the [InterSystems IRIS predefined accounts](#).

By default, all these transactions are unencrypted. You can use TLS to encrypt messages from the browser to the web server. You can use Kerberos to encrypt messages from the gateway to the InterSystems IRIS server as described in [Set Up a Secure Channel for a Web Connection](#); if you are not using Kerberos, you may prefer to physically secure the connection between the host machines, such as by co-locating the gateway and InterSystems IRIS server machines in a locked area with a direct physical connection between them.

Regardless of your architecture, ensure that the data on each connection is secure.

ODBC

InterSystems IRIS supports instance authentication for ODBC connections among all its supported platforms. This requires client-side configuration. The ways of configuring client behavior vary by platform:

- On non-Windows platforms, use the InterSystems ODBC initialization file to specify name-value pairs that provide connection information. This file is described generally in [Using the InterSystems ODBC Driver](#). The file has the following variables relevant to instance authentication:
 - *Authentication Method* — Specifies how the ODBC client authenticates to the DSN. 0 specifies instance authentication; 1 specifies Kerberos.
 - *UID* — Specifies the name for the default user account for connecting to the DSN. At runtime, depending on application behavior, the end-user may be permitted to override this value with a different user account.
 - *Password* — Specifies the password associated with the default user account. If the end-user has been permitted to override the UID value, the application will accept a value for the newly specified user's password.
- On a Windows client, you can specify connection information either through a GUI or programmatically:
 - Through a GUI, there is an ODBC DSN configuration dialog. InterSystems IRIS provides options on the **System DSN** tab. This screen has associated help that describes its fields. The path from the Windows Start menu to display this screen varies by version of Windows; it may be listed in the **Windows Control Panel**, under **Administrative Tools**, on the screen for **Data Sources (ODBC)**.
 - Programmatically, the **SQLDriverConnect** function is available, which accepts a set of name-value pairs. **SQLDriverConnect** is a C call that is part of the ODBC API. Its name-value pairs are the same as those for the initialization file available on non-Windows platforms, except that the password is identified with the *PWD* keyword.

Telnet

When establishing a connection using the InterSystems IRIS Telnet server for Windows, the client uses configuration information that has been stored as part of an InterSystems IRIS remote server. To configure a remote server, go to the client machine. On that machine, the procedure is:

1. Click on the InterSystems IRIS launcher and select **Preferred Server** from the menu (the **Preferred Server** choice also displays the name of the current preferred server).
2. From the submenu that appears, choose **Add/Edit**.
3. To create a new remote server, click the **Add** button; to configure an already-existing server, choose the InterSystems IRIS server to which you are connecting and click the **Edit** button.
4. This displays the **Add Connection** dialog. In the **Authentication Method** area on that dialog, click **Password** for instance authentication.
5. If you are editing the values for an already-existing server, there should be no need to change or add values for the more general fields in this dialog, as they are determined by the server that you chose to edit.

If you are adding a new server, the fields to complete are described in [Define a Remote Server Connection](#).

6. Click **OK** to save the specified values and dismiss the dialog.

Important: When connecting to a non-Windows machine using telnet, there is no InterSystems IRIS telnet server available — you simply use the telnet server that comes with the operating system. Once you have established the connection to the server machine, you can then connect to InterSystems IRIS using the `%Service_Terminal` service.

Delegated Authentication

Delegated Authentication Background

InterSystems IRIS supports *delegated authentication*, which allows you to implement custom mechanisms to replace the authentication and role-management activities that are part of InterSystems security, for example, an enterprise's existing authentication system. As the application developer, you fully control the content of delegated authentication code. Delegated authentication occurs if an instance of InterSystems IRIS finds a **ZAUTHENTICATE** routine in its %SYS namespace. If such a routine exists, InterSystems IRIS uses it to authenticate users, either with calls to new or existing code. InterSystems IRIS includes a routine, ZAUTHENTICATE.mac, that serves as a template for creating the **ZAUTHENTICATE** routine.

Important: If using authentication with HealthShare®, you must use the [ZAUTHENTICATE routine provided by InterSystems](#) and cannot write your own.

How Delegated Authentication Works

When a user attempts to log in and InterSystems IRIS invokes delegated authentication, the sequence of events is:

1. When a service or application uses delegated authentication, a login attempt automatically results in a call to the **ZAUTHENTICATE** routine. The authentication code in this routine can be any user-defined ObjectScript, class methods, or \$ZF callout code.
2. The next step depends on whether or not authentication succeeds and whether or not this is the first login using **ZAUTHENTICATE**:
 - If **ZAUTHENTICATE** succeeds and this is the first time that the user has been authenticated through this mechanism, the user is added to the list of InterSystems IRIS users with a [type](#) of “Delegated user”. If **ZAUTHENTICATE** sets roles or other characteristics, these become part of the user's properties.
 - If **ZAUTHENTICATE** succeeds and this is not the first login, **ZAUTHENTICATE** updates the user's properties.
 - If **ZAUTHENTICATE** fails, then the user receives an access denied error and is unable to access the system. To determine why this has occurred:
 - a. Check the **Reason for Failing to Login** field in the [User Profile](#).
 - b. For more detailed information, check the [audit log](#) for the relevant %System/%Login/LoginFailure event. If auditing or the LoginFailure event are not enabled, you may need to enable both of these and then re-create the circumstances of the login failure.
3. If two-factor authentication is enabled for the instance and the relevant services, then there is a check that the user's PhoneNumber and PhoneProvider properties have been set. If these properties are set, then two-factor authentication proceeds; if they are not set, two-factor authentication cannot proceed and the user is not authenticated.
4. A delegated user is listed as such in the [Type](#) column of the list of users on the **Users** page (**System Administration > Security > Users**). The user's properties are displayed read-only in the Management Portal and are not editable from within InterSystems IRIS (since all the information comes from outside InterSystems IRIS).

Note: A delegated user cannot also be an InterSystems IRIS password user.

Overview of Configuring Delegated Authentication

To use delegated authentication, the steps are:

1. [Create the user-defined authentication code](#) in the **ZAUTHENTICATE** routine. This can include the use of [two-factor authentication](#). This routine can also perform basic setup for a user account, such as specifying roles and other user properties.

If you are using HealthShare Health Connect, create a custom **ZAUTHENTICATE** routine as described in this guide.

If you are using HealthShare Unified Care Record, you cannot create a custom version of **ZAUTHENTICATE** to implement delegated authentication because Unified Care Record comes with its own version of the routine. Instead, you must customize methods in the class `HS.Local.ZAUTHENTICATE`. For more information, see “[Customizing Authentication via Local ZAUTHENTICATE](#)” in the book *Authenticating Users in Unified Care Record*.

2. [Enable delegated authentication](#) for the InterSystems IRIS instance on the [Authentication Options](#) page.
3. Enable delegated authentication for the relevant [services](#), [applications](#), or both, as required.
4. Optionally [enable two-factor authentication](#) for the InterSystems IRIS instance and, if required, for web applications and client-server applications.

For example, to use delegated authentication for an instance’s Management Portal, the steps are:

1. Create the user-defined authentication code in **ZAUTHENTICATE**.
2. Enable delegated authentication for the InterSystems IRIS instance as a whole.
3. Enable delegated authentication for the set of `/csp/sys*` applications.

Create Delegated (User-Defined) Authentication Code

This section describes various aspects of creating your own **ZAUTHENTICATE** routine:

- [Authentication Code Fundamentals](#)
- [Signature](#)
- [Authentication Code](#)
- [Set Values for Roles and Other User Characteristics](#)
- [Return Value and Error Messages](#)

Authentication Code Fundamentals

InterSystems provides a sample routine, `ZAUTHENTICATE.mac`, that you can copy and modify. This routine is part of the Samples-Security sample on GitHub (<https://github.com/interSystems/Samples-Security>). You can download the entire sample as described in [Downloading Samples for Use with InterSystems IRIS](#), but it may be more convenient to simply open the routine on GitHub and copy its contents.

To create your own `ZAUTHENTICATE.mac`:

1. To use `ZAUTHENTICATE.mac` as a template, copy its contents and save them into a `ZAUTHENTICATE.mac` routine in the `%SYS` namespace.
2. Review the comments in the `ZAUTHENTICATE.mac` sample. These provide important guidance about how to implement a custom version of the routine.
3. Edit your routine by adding custom authentication code and any desired code to set user account characteristics.

CAUTION: Because InterSystems IRIS places no constraints on the authentication code in **ZAUTHENTICATE**, the application programmer is responsible for ensuring that this code is sufficiently secure.

Signature

The signature of **ZAUTHENTICATE** is:

ObjectScript

```
ZAUTHENTICATE(ServiceName, Namespace, Username, Password, Credentials,  
              Properties) PUBLIC {  
  
    // authentication code  
    // optional code to specify user account properties and roles  
}
```

where:

- *ServiceName* — A string representing the name of the service through which the user is connecting to InterSystems IRIS, such as **%Service_Console** or **%Service_WebGateway**.
- *Namespace* — A string representing the namespace on the InterSystems IRIS server to which a connection is being established. This is for use with the **%Service_Bindings** service, such as with Studio or ODBC.
- *Username* — A string representing the name of the account entered by the user that is to be validated by the routine's code.
- *Password* — A string representing the password entered by the user that is to be validated.
- *Credentials* — *Passed by reference*. Not implemented in this version of InterSystems IRIS.
- *Properties* — *Passed by reference*. An array of returned values that defines characteristics of the account named by *Username*.

When InterSystems IRIS calls **ZAUTHENTICATE**, it has values for these arguments and supplies them to the routine.

Note: In older versions of InterSystems products, **ZAUTHENTICATE** took four arguments. For backwards compatibility, you can still use the four-argument version. If you are updating your code from the old to new version, note that the new arguments are second and fifth ones: *Namespace* and *Credentials*.

Authentication Code

The content of authentication code is application specific. If authentication succeeds, the routine should return the `$$$OK` macro; otherwise, it should return an error code. See [Return Value and Error Messages](#) for more information on return values.

CAUTION: Because InterSystems IRIS does not and cannot place any constraints on the authentication code in **ZAUTHENTICATE**, the application programmer is responsible for ensuring that this code is sufficiently secure.

The GetCredentials Entry Point

ZAUTHENTICATE includes an **GetCredentials** entry point. This entry point is called whenever delegated authentication is enabled for a service, and is called before the user is prompted for a username and password. Instead of getting a username and password from the user, code in the function (created by the application developer) specifies the username and password. The username and password returned are then authenticated in the normal manner as if the user entered them. A possible use of this mechanism is to provide a username and password within the entry point and then, within authentication code, to \$roles for the process.

The username and password returned from this entry point can be obtained by any mechanism that the application developer chooses. They can come from a global, come from an external DLL or LDAP call, or simply be set within the routine. The application developer could even provide code to prompt for the username and password, such as in a terminal connection or with a login page.

When there is a call to the **GetCredentials** entry point, the return value and other factors determine what happens next:

- If the code sets the values of *Username* and *Password* and also returns a success status (\$\$\$OK), then:
 - There is no additional username/password prompting.
 - The authentication process proceeds.

Important: If the access point returns \$\$\$OK, then its code must set the values of *Username* and *Password*. Otherwise, the user is denied access to the system and an error is written to the audit log.

- If the entry point returns the error status \$SYSTEM.Status.Error(\$\$\$GetCredentialsFailed), then normal username/password prompting proceeds.
- If the entry point returns any other error status, then:
 - The user is denied access to the system.
 - The error is logged in the audit log.

In the following example of a **GetCredentials** entry point, the code performs different actions for different services:

- For **%Service_Console**, it does not prompt the user for any information and sets the process's username and password to `_SYSTEM` and `SYS`, respectively.
- For **%Service_Bindings**, it forces the user to provide a username and password.
- For web applications, it checks if the application in use is the `/csp/` samples application; if it is that application, it sets the username and password to `AdminUser` and `Test`. For all other web applications, it denies access.
- For any other service, it denies access.

Finally, the **Error** entry point performs clean-up as necessary.

The code is:

ObjectScript

```
GetCredentials(ServiceName, Namespace, Username, Password, Credentials) Public {

    // For console sessions, authenticate as _SYSTEM.
    If ServiceName="%Service_Console" {
        Set Username="_SYSTEM"
        Set Password="SYS"
        Quit $SYSTEM.Status.OK()
    }

    // For a web application, authenticate as AdminUser.
    If $isobject($get(%request)) {
        If %request.Application="/csp/samples/" {
            Set Username="AdminUser"
            Set Password="Test"
            Quit $System.Status.OK()
        }
    }

    // For bindings connections, use regular prompting.
    If ServiceName="%Service_Bindings" {
        Quit $SYSTEM.Status.Error($$$GetCredentialsFailed)
    }

    // For all other connections, deny access.
    Quit $SYSTEM.Status.Error($$$AccessDenied)
}
```

For more details, see the comments for this entry point in **ZAUTHENTICATE.mac**.

The `SendTwoFactorToken` Entry Point

ZAUTHENTICATE includes an **SendTwoFactorToken** entry point. This entry point is for use with two-factor authentication. If it is defined and the InterSystems IRIS instance has two-factor authentication enabled, then you can override the default system setting for the format of the message and token that the instance sends to the user's mobile phone. This allows for messages that can vary by application even on the same InterSystems IRIS instance.

For more details and an example of how to use this entry point, see this entry point in the sample **ZAUTHENTICATE.mac**.

Set Values for Roles and Other User Characteristics

If initial authentication succeeds, **ZAUTHENTICATE** can establish the roles and other characteristics for the authenticated user. For subsequent logins, **ZAUTHENTICATE** can update these elements of the user record.

For this to happen, code in **ZAUTHENTICATE** sets the values of the *Properties* array. (*Properties* is passed by reference to **ZAUTHENTICATE**.) Typically, the source for the values being set is a repository of user information that is available to **ZAUTHENTICATE**.

User Properties

The elements in the *Properties* array are:

- *Properties("Comment")* — Any text
- *Properties("FullName")* — The first and last name of the user
- *Properties("NameSpace")* — The default namespace for a Terminal login
- *Properties("Roles")* — The comma-separated list of roles that the user holds in InterSystems IRIS
- *Properties("Routine")* — The routine that is executed for a Terminal login
- *Properties("Password")* — The user's password
- *Properties("Username")* — The user's username
- *Properties("PhoneNumber")* — The user's mobile phone number, for use with two-factor authentication
- *Properties("PhoneProvider")* — The user's mobile phone's service provider, for use with two-factor authentication

Each of these elements is described in more detail in one of the following sections.

Note: The value of each element in the properties array determines the value of its associated property for the user being authenticated. It is not possible to use only a subset of the properties or to manipulate their values after authentication.

Comment

If **ZAUTHENTICATE** sets the value of *Properties("Comment")*, then that string becomes the value of the user account's *Comment* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) If no value is passed back to the calling routine, then the value of *Comment* for the user account is a null string and the relevant field in the Management Portal then holds no content.

FullName

If **ZAUTHENTICATE** sets the value of *Properties("FullName")*, then that string becomes the value of the user account's *Full name* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) If no value is passed back to the calling routine, then the value of *Full name* for the user account is a null string and the relevant field in the Management Portal then holds no content.

Namespace

If **ZAUTHENTICATE** sets the value of *Properties("Namespace")*, then that string becomes the value of the user account's *Startup Namespace* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) If no value is passed back to the calling routine, then the value of *Startup Namespace* for the user account is a null string and the relevant field in the Management Portal then holds no content.

Once connected to InterSystems IRIS, the value of *Startup Namespace* (hence, that of *Properties("Namespace")*) determines the initial namespace for any user authenticated for local access (such as for Console, Terminal, or Telnet). If *Startup Namespace* has no value (since *Properties("Namespace")* has no value), then the initial namespace for any user authenticated for local access is determined as follows:

1. If the USER namespace exists, that is the initial namespace.
2. If the USER namespace does not exist, the initial namespace is the %SYS namespace.

Note: If the user does not have the appropriate privileges for the initial namespace, access is denied.

Password

If **ZAUTHENTICATE** sets the value of *Properties("Password")*, then that string becomes the value of the user account's *Password* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) If no value is passed back to the calling routine, then the value of *Password* for the user account is a null string and the relevant field in the Management Portal then holds no content.

Roles

If **ZAUTHENTICATE** sets the value of *Properties("Roles")*, then that string specifies the *Roles* to which a user is assigned; this value is a string containing a comma-delimited list of roles. If no value is passed back to the calling routine, then the value of *Roles* for the user account is a null string and the relevant field in the Management Portal then holds no content. Information about a user's [roles](#) is available on the **Roles** tab of a user's **Edit User** page.

If any roles returned in *Properties("Roles")* are not defined, then the user is not assigned to the role.

Hence, the logged-in user is assigned to roles as follows:

- If a role is listed in *Properties("Roles")* and is defined by the InterSystems IRIS instance, then the user is assigned to the role.
- If a role is listed in *Properties("Roles")* and is not defined by the InterSystems IRIS instance, then the user is not assigned to the role.
- A user is always assigned to those roles associated with the _PUBLIC user. A user also has access to all public resources. For information on the _PUBLIC user, see [The _PUBLIC Account](#); for information on public resources, see [Services and Their Resources](#).

Routine

If **ZAUTHENTICATE** sets the value of *Properties("Routine")*, then that string becomes the value of the user account's *Startup Tag^Routine* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) If no value is passed back to the calling routine, then the value of *Startup Tag^Routine* for the user account is a null string and the relevant field in the Management Portal then holds no content.

If *Properties("Routine")* has a value, then this value specifies the routine to execute automatically following login on a terminal-type service (such as for Console, Terminal, or Telnet). If *Properties("Routine")* has no value, then login starts the Terminal session in programmer mode.

Username

If **ZAUTHENTICATE** returns the *Username* property, then the value of *Username* is written to the security database after any processing in the function; this provides chance to modify the value that the user entered at the prompt. If **ZAUTHENTICATE** does not return the *Username* property, then the value of the property is written to the security database as entered.

If **ZAUTHENTICATE** sets the value of *Properties("Username")*, then that string becomes the value of the user account's *Name* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) This provides the application programmer with an opportunity to normalize content provided by the end-user at the login prompt.

If there is no explicit call that passes the value of *Properties("Username")* back to the calling routine, then there is no normalization and the value entered by the end-user at the prompt serves as the value of the user account's *Name* property without any modification.

PhoneNumber and PhoneProvider

These are properties associated with [two-factor authentication](#).

If **ZAUTHENTICATE** sets the value of *Properties("PhoneNumber")* and *Properties("PhoneProvider")*, then these are then written to the InterSystems IRIS database for the user as the user's mobile phone number and mobile phone service provider. If these are not passed back to the calling routine, then the phone number and service provider written to the InterSystems IRIS database are a null string. Hence, to use two-factor authentication with delegated authentication, you must supply both of these.

The User Information Repository

ZAUTHENTICATE can refer to any kind of repository of user information, such as a global or an external file. It is up to the code in the routine to set any external properties in the *Properties* array so that the authenticated user can be created or updated with this information. For example, while a repository can include information such as roles and namespaces, **ZAUTHENTICATE** code must make that information available to InterSystems IRIS.

If information in the repository changes, this information is only propagated back into the InterSystems IRIS user information if there is code in **ZAUTHENTICATE** to perform this action. Also, if there is such code, changes to users' roles must occur in the repository; if you change a user's roles during a session, the change does not become effective until the next login, at which point the user's roles are re-set by **ZAUTHENTICATE**.

Return Value and Error Messages

The routine returns one of the following values:

- Success — `$$$OK`. This indicates that username/password combination was successfully authenticated
- Failure — `$$SYSTEM.Status.Error($$$ERRORMESSAGE)`. This indicates that authentication failed.

ZAUTHENTICATE can return system-defined or application-specific error messages. All these messages use the **Error** method of the `%SYSTEM.Status` class. This method is invoked as `$$SYSTEM.Status.Error` and takes one or two arguments, depending on the error condition.

The available system-defined error messages are:

- `$$SYSTEM.Status.Error($$$AccessDenied)` — Error message of "Access Denied"
- `$$SYSTEM.Status.Error($$$InvalidUsernameOrPassword)` — Error message of "Invalid Username or Password"
- `$$SYSTEM.Status.Error($$$UserNotAuthorizedOnSystem,Username)` — Error message of "User *Username* is not authorized"
- `$$SYSTEM.Status.Error($$$UserAccountIsDisabled,Username)` — Error message of "User *Username* account is disabled"

- **`$$SYSTEM.Status.Error($$$UserInvalidUsernameOrPassword,Username)`** — Error message of “User *Username* invalid name or password”
- **`$$SYSTEM.Status.Error($$$UserLoginTimeout)`** — Error message of “Login timeout”
- **`$$SYSTEM.Status.Error($$$UserCTRLC)`** — Error message of “Login aborted”
- **`$$SYSTEM.Status.Error($$$UserDoesNotExist,Username)`** — Error message of “User *Username* does not exist”
- **`$$SYSTEM.Status.Error($$$UserInvalid,Username)`** — Error message of “Username *Username* is invalid”
- **`$$SYSTEM.Status.Error($$$PasswordChangeRequired)`** — Error message of “Password change required”
- **`$$SYSTEM.Status.Error($$$UserAccountIsExpired,Username)`** — Error message of “User *Username* account has expired”
- **`$$SYSTEM.Status.Error($$$UserAccountIsInactive,Username)`** — Error message of “User *Username* account is inactive”
- **`$$SYSTEM.Status.Error($$$UserInvalidPassword)`** — Error message of “Invalid password”
- **`$$SYSTEM.Status.Error($$$ServiceDisabled,ServiceName)`** — Error message of “Logins for Service *ServiceName* are disabled”
- **`$$SYSTEM.Status.Error($$$ServiceLoginsDisabled)`** — Error message of “Logins are disabled”
- **`$$SYSTEM.Status.Error($$$ServiceNotAuthorized,ServiceName)`** — Error message of “User not authorized for service”

To generate a custom message, use the **`$$SYSTEM.Status.Error()`** method, passing it the **`$$$GeneralError`** macro and specifying any custom text as the second argument. For example:

```
$$SYSTEM.Status.Error($$$GeneralError,"Any text here")
```

Note that when an error message is returned to the caller, it is logged in the audit database (if LoginFailure event auditing is turned on). However, the only error message the user sees is **`$$SYSTEM.Status.Error($$$AccessDenied)`**. However, the user also sees the message for the **`$$$PasswordChangeRequired`** error. Return this error if you want the user to change from the current to a new password.

Set Up Delegated Authentication

Once you have created a **ZAUTENTICATE** routine to perform authentication (and, optionally, authorization tasks), the next step is to enable it for the instance’s relevant services or applications. This procedure is:

1. Enable delegated authentication for entire instance. On the **Authentication/Web Session Options** page (**System Administration > Security > System Security > Authentication/Web Session Options**), select **Allow Delegated authentication** and click **Save**.

With delegated authentication enabled for the instance, a **Delegated** check box appears on the **Edit Service** page for relevant services and the **Edit Web Application** page for those applications.

2. Enable delegated authenticated for services and applications, as appropriate.

The following services support delegated authentication:

- **`%Service_Bindings`**
- **`%Service_CallIn`**
- **`%Service_ComPort`**
- **`%Service_Console`**
- **`%Service_Login`**

- `%Service_Terminal`
- `%Service_Telnet`
- `%Service_WebGateway`

These fall into several categories of access modes:

- [Local Access](#) —

`%Service_CallIn`, `%Service_ComPort`, `%Service_Console`, `%Service_Login`, `%Service_Terminal`, `%Service_Telnet`

To use delegated authentication with local connections, enable it for the service.

- [Client-Server Access](#) —

`%Service_Bindings`

To use delegated authentication with client-server connections, enable it for the service.

- [Web Access](#) —

`%Service_WebGateway`

To use delegated authentication with web-based connections, enable it for the web application. You may also enable it for the web gateway by enabling the service `%Service_WebGateway`

After Delegated Authentication Succeeds

Once the user has authenticated, two important topics are:

- [The State of the System](#)
- [Change Passwords](#)

The State of the System

Any user who is initially authenticated using delegated authentication is listed in the table of users on the **Users** page (**System Administration > Security > Users**) as having a type of “Delegated user”. If a system administrator has explicitly created a user through the Management Portal (or using any other native InterSystems IRIS facility), that user has a type of “InterSystems IRIS password user”. If a user attempts to log in using delegated authentication and is successfully authenticated, InterSystems IRIS determines that this user already exists as an InterSystems IRIS user — not a Delegated user — and so login fails.

Note: To perform sharded operations on a sharded cluster, a delegated user must have been previously authenticated on each node of the sharded cluster by some means other than an internal sharding connection. For more information on sharding, see [Horizontally Scaling InterSystems IRIS with Sharding](#).

Change Passwords

The `ZAUTHENTICATE` routine also includes an entry point, `ChangePassword`, to include code to change a user’s password. The signature of this entry point is:

ObjectScript

```
ChangePassword(Username,NewPassword,OldPassword,Status) Public {}
```

where

- *Username* is a string specifying the user whose password is being changed.

- *NewPassword* is a string specifying the new value of the user's password.
- *OldPassword* is a string specifying the old value of the user's password.
- *Status* (passed by reference) receives an InterSystems IRIS status value indicating either that the password change has been successful or specifying the error that caused the routine to fail.

Use LDAP with Delegated Authentication or Other Mechanisms

You can also use LDAP as part of a custom authentication system (that is, with the InterSystems IRIS delegated authentication feature). To do this, use calls to the %SYS.LDAP class as part of the custom authentication code in the **ZAUTHENTICATE** routine.

InterSystems provides a sample routine, LDAP.mac, that demonstrates these calls. This routine is part of the Samples-Security sample on GitHub (<https://github.com/interSystems/Samples-Security>).

Also, if you need to authenticate to LDAP or use instance authentication after collecting credentials through another mechanism, call **\$SYSTEM.Security.Login** with those credentials to authenticate the user.

Two-Factor Authentication

Two-Factor Authentication

In addition to the authentication mechanism in use, InterSystems IRIS supports the use of *two-factor authentication*. This means that InterSystems authentication can require the end-user to possess two separate elements or “factors.” From the end-user’s perspective, the first factor is something that you know — for example, a password; the second factor is something that you have — for example, a smart phone. InterSystems IRIS performs two-factor authentication on its end-users using either of two mechanisms:

- *SMS text authentication* — InterSystems IRIS sends a security code to the end-user’s phone via SMS. The end-user enters that code when prompted.
- *Time-based one-time password (TOTP)* — The end-user initially receives a secret key from InterSystems IRIS. That key is a *shared secret* between InterSystems IRIS and the end-user’s application (such as an app on a mobile phone) or physical authentication device; both use the key and other information to generate a TOTP that serves as a verification code and that the end-user enters at an InterSystems IRIS prompt. The TOTP expires after 60 seconds and the end-user can only use it a single time, which is why it is called *time-based* and *one-time*.

This section covers the following topics:

- [Overview of Setting Up Two-Factor Authentication](#)
- [Configure Two-Factor Authentication for the Server](#)
- [Enable or Disable Two-Factor Authentication for a Service](#)
- [Configure Web Applications for Two-Factor Authentication](#)
- [Configure an End-User for Two-Factor Authentication](#)
- [Configure Bindings Clients for Two-Factor Authentication](#)

Overview of Setting Up Two-Factor Authentication

The major steps to setting up two-factor authentication are:

1. [Enable and configure two-factor authentication for the instance as a whole](#). You can configure the instance to use SMS text authentication, TOTP authentication, or both. For details about TOTP authentication, see [Two-factor TOTP overview](#).
2. For SMS text authentication, [configure the mobile phone service provider\(s\)](#), if necessary. This includes:
 - Adding any mobile phone service providers if any are required and are not included in the list of default providers.
 - Changing configuration information as necessary for any existing providers (default or added).
3. Configure the service, as appropriate:
 - `%Service_Bindings` — [Enable two-factor authentication for the service](#) and continue to the next step.
 - `%Service_Console` and `%Service_Terminal` — Simply [enable two-factor authentication for the service](#). This is all that is required.
 - `%Service_WebGateway` — There is no central means of enabling two-factor authentication for `%Service_WebGateway`. Continue to the next step.

You can enable either or both types of authentication for each service. For more information about services, see [Services](#).

4. Configure client-server applications and web applications, as appropriate:

- a. For client-server applications (those that use `%Service_Bindings`), [add the appropriate calls into the client application to support it](#); this is a programming task that varies according to the client-side component in use (for example, Java, JDBC, or .NET, among others).

Important: Two-factor authentication is designed to receive a response from a human end-user in real time. If what the end-user considers a single session actually consists of multiple, sequential sessions, then the repeated prompting for the second factor may result in an unexpectedly difficult user experience. With client-server applications, the underlying protocol often causes clients to establish, disconnect, and reestablish connections repeatedly; such activity makes the use of two-factor authentication less desirable for this type of application.

- b. For web applications (those that use `%Service_WebGateway`), [configure each application to support it](#).

Note: For the InterSystems IRIS [Terminal](#), which uses the `%Service_Console` service on Windows and the `%Service_Terminal` service on other operating systems, there is no configuration required other than server-side setup; since InterSystems IRIS controls the prompting in these, it simply follows the standard prompt (regardless of the authentication mechanism) with the two-factor authentication prompt and processes end-user input accordingly.

5. If you are using delegated authentication, modify the `ZAUTHENTICATE.mac` routine as required. See [Delegated Authentication](#) for more information.
6. [Configure each end-user to enable SMS text authentication or TOTP authentication](#). An end-user can be configured to use both mechanisms, but cannot have both mechanisms enabled simultaneously.

Two-Factor TOTP Overview

Two-factor authentication using a time-based one-time password (TOTP) authentication works as follows:

1. Select either an authentication device or an application that generates a TOTP, and then provide it or ensure that your users have it.
2. When you configure an end-user for two-factor TOTP authentication, the system generates a secret key, which is displayed as a base-32 encoded randomized bit string. InterSystems IRIS and the end-user share this secret key (which is why it is known as a *shared secret*). Both InterSystems IRIS and the end-user's authentication device or application use it to generate the TOTP itself, which serves as a verification code. The TOTP, which the end-user enters into a **Verification code** field or prompt, is a string of six digits, and a new one is generated at a regular interval (thirty seconds, by default).
3. At login time, after the end-user provides InterSystems IRIS with a password, InterSystems IRIS then additionally prompts for the TOTP. The end-user provides the TOTP, and then completes the login process.

The end-user can get the secret key from InterSystems IRIS in several ways:

- When you configure the end-user's account to support two-factor TOTP authentication, the **Edit User** page for the end-user displays the end-user's secret key, as well as the name of the issuer and the end-user's account name. It also displays a QR code that includes all this information (a QR code is a machine-readable code such as the one pictured below). The end-user can then enter the information into an authentication device or an application by scanning the code or entering the information manually.
- If you choose to show the end-user their secret key during the login to a web application or Terminal session (using `%Service_Console` or `%Service_Terminal`), you can enable this behavior by selecting the **Display Time-Based One-time Password QR Code on next login** field on the **Edit User** page. The Terminal session will then display the end-user's issuer, account, and secret key. A web application will display the end-user's issuer, account, and secret key, along with a QR code; here, the end-user can then scan the code or enter the information manually.

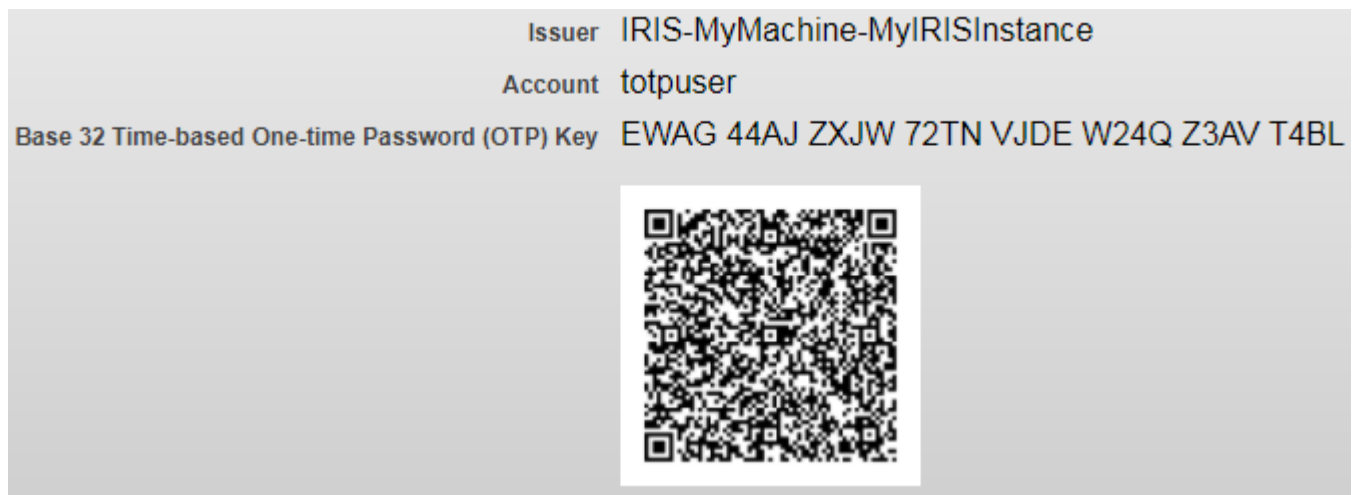
Important: InterSystems does not recommend this option. See the following caution for more details.

CAUTION: The following are critical security concerns when using two-factor TOTP authentication:

- Do *not* transmit the secret key or QR code in an unsecured environment. Out-of-band transmission is preferable to transmission even on a secure network. (The secret key gives an end-user the means to log in to InterSystems IRIS or an InterSystems IRIS application. If you and your end-users do not ensure the secret key's safety, then an attacker may gain access to it, which renders it useless for security.)
- When configuring two-factor TOTP authentication for your organization, InterSystems strongly recommends that you provide the secret key to each end-user in person or by phone, or that you have the end-user scan the QR code in the physical presence of an administrator. This provides the opportunity to authenticate the individual who obtains the secret key.

Delivering the secret key over the network increases the possibility of exposing it. This includes displaying the secret key to the end-user when they first log in to a web application, console, or the Terminal; this also includes displaying the QR code to the end-user when they first log in to a web application.

Figure B–2: A TOTP Issuer, Account, Key, and QR Code



Note: If you are using two-factor TOTP authentication and wish to generate QR codes, Java 1.7 or higher must be running on the InterSystems IRIS server. Without Java, InterSystems IRIS can use two-factor TOTP authentication, but the end-user enters the values for the issuer, account, and key manually on the authentication device or in the application.

Configure Two-Factor Authentication for the Server

The steps in configuring two-factor authentication for the InterSystems IRIS server are:

1. [Enable and configure two-factor authentication for the instance as a whole](#). You can configure the instance to use SMS text authentication, TOTP authentication, or both.
2. For SMS text authentication, [configure the mobile phone service provider\(s\)](#), if necessary. This includes:
 - Adding any mobile phone service providers if any are required and are not included in the list of default providers.
 - Changing configuration information as necessary for any existing providers (default or added).

Enable and Configure Two-Factor Authentication Settings for an Instance

When setting up two-factor authentication for an InterSystems IRIS instance (server), you can enable one or both of:

- **Two-factor time-based one-time password authentication** (TOTP authentication)
- **Two-factor SMS text authentication**

To enable either form of two-factor authentication, the procedure is:

1. From the Management Portal home page, go to the **Authentication/Web Session Options** page (**System Administration** > **Security** > **System Security** > **Authentication/Web Session Options**).
2. To enable two-factor TOTP authentication, on the **Authentication/Web Session Options** page, select the **Allow Two-Factor Time-Based One-Time Password Authentication** check box. This displays the **Two-Factor Time-Based One-Time Password Issuer** field; here, enter a string to identify this instance of InterSystems IRIS.
3. To enable two-factor SMS text authentication, on the **Authentication/Web Session Options** page, select the **Allow Two-Factor SMS Text Authentication** check box. This displays the following fields:
 - **Two-Factor Timeout (secs)** — Optional timeout in seconds for entering the one-time security token.
 - **DNS name of SMTP server** — The DNS (Domain Name Service) name of the SMTP (Simple Mail Transfer Protocol) server that this instance of InterSystems IRIS is using to send SMS text messages, such as smtp.example.com (required).
 - **From (address)** — Address to appear in the “From” field of message (required).
 - **SMTP username** — Optional username for SMTP authentication (if the SMTP server requires it).
 - **SMTP Password** and **SMTP Password (confirm)** — Optional password (entered and confirmed) for SMTP authentication (if the SMTP server requires it).
4. Click **Save**.
5. If the instance is supporting SMS text authentication, configure mobile phone service providers as required. These procedures are described in the next section.

After completing this process for the instance itself, you may need to perform other configuration, such as for the instance’s services, web applications, and client-server applications; you *will* need to configure the instance’s users. [The Overview of Setting Up Two-Factor Authentication](#) provides general direction about this.

Configure Mobile Phone Service Providers

The topics related to configuring mobile phone service providers are:

- [Create or Edit a Mobile Phone Service Provider](#)
- [Delete a Mobile Phone Service Provider](#)
- [Predefined Mobile Phone Service Providers](#)

Create or Edit a Mobile Phone Service Provider

To create or edit a mobile phone service provider, the procedure is:

1. From the Management Portal home page, go to the **Mobile Phone Service Providers** page (**System Administration** > **Security** > **Mobile Phone**):
 - To create a new provider, click **Create New Provider**.
 - To edit an existing provider, click **Edit** on the provider’s row in the table of providers.

This displays the **Edit Phone Provider** page for the selected mobile phone service provider.

2. On the **Edit Phone Provider** page, enter or change the value for each of the following fields:
 - **Service Provider** — The name of the mobile phone service provider (typically, its company name).
 - **SMS Gateway** — The address of the server that the mobile phone service provider uses to dispatch SMS (short message service) messages.

Delete a Mobile Phone Service Provider

To delete a mobile phone service provider, the procedure is:

1. From the Management Portal home page, go to the **Mobile Phone Service Providers** page (**System Administration > Security > Mobile Phone**).
2. On the **Mobile Phone Service Providers** page, in the row of the provider, click **Delete**.
3. When prompted to confirm the deletion, click **OK**.

Predefined Mobile Phone Service Providers

InterSystems IRIS ships with a predefined list of mobile phone service providers, each with its SMS (short message service) gateway preset. These are:

- AT&T Wireless — txt.att.net
- Alltel — message.alltel.com
- Cellular One — mobile.celloneusa.com
- Nextel — messaging.nextel.com
- Sprint PCS — messaging.sprintpcs.com
- T-Mobile — tmomail.net
- Verizon — vtext.com

Enable or Disable Two-Factor Authentication for a Service

Important: For `%Service_WebGateway`, there is no central location for enabling or disabling two-factor authentication. Enable or disable it for each application as described in [Configure Web Applications for Two-Factor Authentication](#).

To enable or disable two-factor authentication for `%Service_Bindings`, `%Service_Console`, and `%Service_Terminal`, procedure is:

1. From the Management Portal home page, go to the **Services** page (**System Administration > Security > Services**).
2. On the **Services** page, click the name of the service for which you wish to enable either form of two-factor authentication. This displays the **Edit Service** page for the service.
3. On the service's **Edit Service** page, select or clear the **Two-factor SMS** check box, **Two-factor Time-based One-time Password** check box, or both. Note that each of these check boxes only appear if two-factor authentication is enabled for the instance.
4. Click **Save**.

Configure Web Applications for Two-Factor Authentication

Once you have enabled two-factor authentication for an instance, you must enable it for all web applications that will use it. The procedure to enable it for an application is:

1. From the Management Portal home page, go to the **Web Applications** page (**System Administration > Security > Applications > Web Applications**).
2. On the **Web Applications** page, for the application you wish to enable two-factor authentication, click the name of the application, which displays its **Edit** page.
3. On the **Edit** page, in the **Security Settings** section of the page, select or clear the **Two-factor SMS** check box, **Two-factor Time-based One-time Password** check box, or both. Note that each of these check boxes only appear if two-factor authentication is enabled for the instance.

Note: A web application cannot simultaneously support both two-factor authentication and web services.

Configure an End-User for Two-Factor Authentication

To configure an end-user to receive a one-time security token for two-factor authentication, the procedure is:

1. From the Management Portal home page, go to the **Users** page (**System Administration > Security > Users**):
2. For an existing user, click the name of the user to edit; for a new user, begin creating the user by clicking **Create New User** (for details about creating a new user, see [Create a New User](#)). Either of these actions displays the **Edit** page for the end-user.
3. On the **Edit User** page, select **SMS text enabled** or **Time-based One-time Password enabled**, as appropriate.
4. If you select **SMS Text**, you must complete the following fields:
 - **Mobile phone service provider** — The company that provides mobile phone service for the user. Either select a provider from those listed or, if the provider does not appear in the list, click **Create new provider** to add a new provider for the InterSystems IRIS instance. (Clicking **Create a new provider** displays the **Create a New Mobile Phone Provider** window, which has fields for the **Service Provider** and the **SMS Gateway**, the purpose of which are identical to those described in [Create or Edit a Mobile Phone Service Provider](#).)
 - **Mobile phone number** — The user's mobile phone number. This is the second factor, and is where the user receives the text message containing the one-time security token.
5. If you select **Time-based One-time Password enabled**, the page displays the following fields and information:
 - **Display Time-Based One-time Password QR Code on next login** — Whether or not to display a QR code when the user next logs in. If selected, InterSystems IRIS displays the code at the next login and prompts the user to scan it into the authentication device or application, and then to provide the displayed token to complete the authentication process. By default, this option is not selected. InterSystems recommends that you do *not* use this option.
 - **Generate a new Time-based One-time Password Key** — Creates and displays both a new shared secret for the end-user and a new QR code.

Important: If you generate a new time-based one-time password key for a user, the current key in the user's authenticator application will no longer work. Before logging in, the user must enter the new key into the authenticator, either by scanning the QR code or by manually entering it. (This does not affect existing sessions.)
 - **Issuer** — The identifier for the InterSystems IRIS instance, which you established when configuring two-factor TOTP authentication for the instance.
 - **Account** — The identifier for the InterSystems IRIS account, which is the account's username.

- **Base-32 Time-Based One-Time Password (OTP) Key** — The secret key that the end-user enters into the authentication device or application.
- **QR Code** — A scannable code that contains the values of the issuer, account, and secret key.

6. Click **Save** to save these values for the user.

If a service uses two-factor authentication and an end-user has two-factor authentication enabled, then authentication requires:

- For SMS text authentication, a mobile phone that is able to receive text messages on that phone.
- For TOTP authentication, an application or authentication device that can generate verification codes.

Otherwise, the end-user cannot authenticate:

- For SMS text authentication, the end-user must have a mobile phone and be able to receive text messages on that phone. This is the phone number at which the user receives a text message containing the one-time security token as an SMS text.
- For TOTP authentication, the user must have an authentication device or application that can either scan a QR code or that can accept the secret key and other information required to generate each TOTP (which serves as a verification code).

Configure Bindings Clients for Two-Factor Authentication

Client-server connections use `%Service_Bindings`. For these connections, the code required to use two-factor authentication varies by programming language. (Note that Console, the Terminal, and web applications do not require any client-side configuration.) Supported languages include:

- [Java and JDBC](#)
- [.NET](#)
- [ODBC](#)

Client-side code performs three operations:

1. After establishing a connection to the InterSystems IRIS server, it checks if two-factor authentication is enabled on the server. Typically, this uses a method of the client's connection object.
2. It gets the one-time security token from the user. This generally involves user-interface code that is not specifically related to InterSystems IRIS.
3. It provides the one-time security token to the InterSystems IRIS server. This also typically uses a connection object method.

Note: When a user logs in through `%Service_Bindings`, InterSystems IRIS does not present a QR code to scan. The user must have previously set up the authentication device or application.

Important: Studio, which connects to the InterSystems IRIS server using `%Service_Bindings`, does not support two-factor authentication.

Java and JDBC

With Java, support for two-factor authentication uses two methods of the `IRISConnection` class:

- `public boolean isTwoFactorEnabled() throws Exception`

This method checks if two-factor authentication is enabled on the server. It returns a boolean; `true` means that two-factor authentication is enabled.

- `public void sendTwoFactorToken(String token) throws Exception`

This method provides the one-time security token to the server. It takes one argument, *token*, the one-time security token that the user has received.

The following example uses an instance of a connection called *conn*:

1. It uses that instance's methods to check if two-factor authentication is enabled.
2. It attempts to provide the token to the server and performs error processing if this fails.

Important: If two-factor authentication is enabled on the server and the client code does not implement two-factor authentication calls, then the server will drop the connection with the client.

```
// Given a connection called "conn"
if (conn.isTwoFactorEnabled()) {
    // Prompt the user for the two-factor authentication token.
    // Store the token in the "token" variable.
    try {
        conn.sendTwoFactorToken(token);
    }
    catch (Exception ex) {
        // Process the error from a invalid authentication token here.
    }
}
```

.NET

For .NET, InterSystems IRIS supports connections with two-factor authentication with the managed provider and with ADO.NET. Support for two-factor authentication uses two methods of the `tcp_conn` class:

- `bool IRISConnection.isTwoFactorEnabledOpen()`

This method opens a connection to the InterSystems IRIS server and checks if two-factor authentication is enabled there. It returns a boolean; `true` means that two-factor authentication is enabled.

- `void IRISConnection.sendTwoFactorToken(token)`

This method provides the one-time security token to the server. It has no return value. It takes one argument, *token*, the one-time security token that the user has received. If there is a problem with either the token (such as if it is not valid) or the connection, then the method throws an exception.

Important: A client application makes a call to **isTwoFactorEnabledOpen** *instead of* a call to **IRISConnection.Open**. The **isTwoFactorEnabledOpen** method requires a subsequent call to **sendTwoFactorToken**.

Also, if two-factor authentication is enabled on the server and the client code does not implement two-factor authentication calls, then the server will drop the connection with the client.

The following example uses an instance of a connection called *conn*:

1. It uses that instance's methods to check if two-factor authentication is enabled.

2. It attempts to provide the token to the server and performs error processing if this fails.

```
// Given a connection called "conn"
try {
    if (conn.isTwoFactorEnabledOpen()) {
        // Prompt the user for the two-factor authentication token.
        // Store the token in the "token" variable.
        conn.sendTwoFactorToken(token);
    }
}
catch (Exception ex) {
    // Process exception
}
```

ODBC

With ODBC, support for two-factor authentication uses two standard ODBC function calls (which are documented in the [Microsoft ODBC API Reference](#)):

- `SQLRETURN rc = SQLGetConnectAttr(conn, 1002, &attr, sizeof(attr), &stringLengthPtr);`

The [SQLGetConnectAttr](#) function, part of the Microsoft ODBC API, returns the current value of a specified connection attribute. The InterSystems ODBC client uses this function to determine if the server supports two-factor authentication. The value of the first argument is a handle to the connection from the client to the server; the value of the second argument is 1002, the ODBC attribute that specifies whether or not two-factor authentication is supported; the values of the subsequent arguments are for the string containing the value of attribute 1002, as well as relevant variable sizes.

- `SQLRETURN rc = SQLSetConnectAttr(conn, 1002, securityToken, SQL_NTS);`

The [SQLSetConnectAttr](#) function, also part of the Microsoft ODBC API, sets the value of a specified connection attribute. The InterSystems ODBC client uses this function to send the value of the two-factor authentication token to the server. The values of the four arguments are, respectively:

- The connection from the client to the server.
- 1002, the ODBC attribute that specifies whether or not two-factor authentication is supported.
- The value of the one-time security token.
- `SQLNTS`, which indicates that the one-time security token is stored in a string.

Important: If two-factor authentication is enabled on the server and the client code does not implement two-factor authentication calls, then the server will drop the connection with the client.

The following example uses an instance of a connection called *conn*:

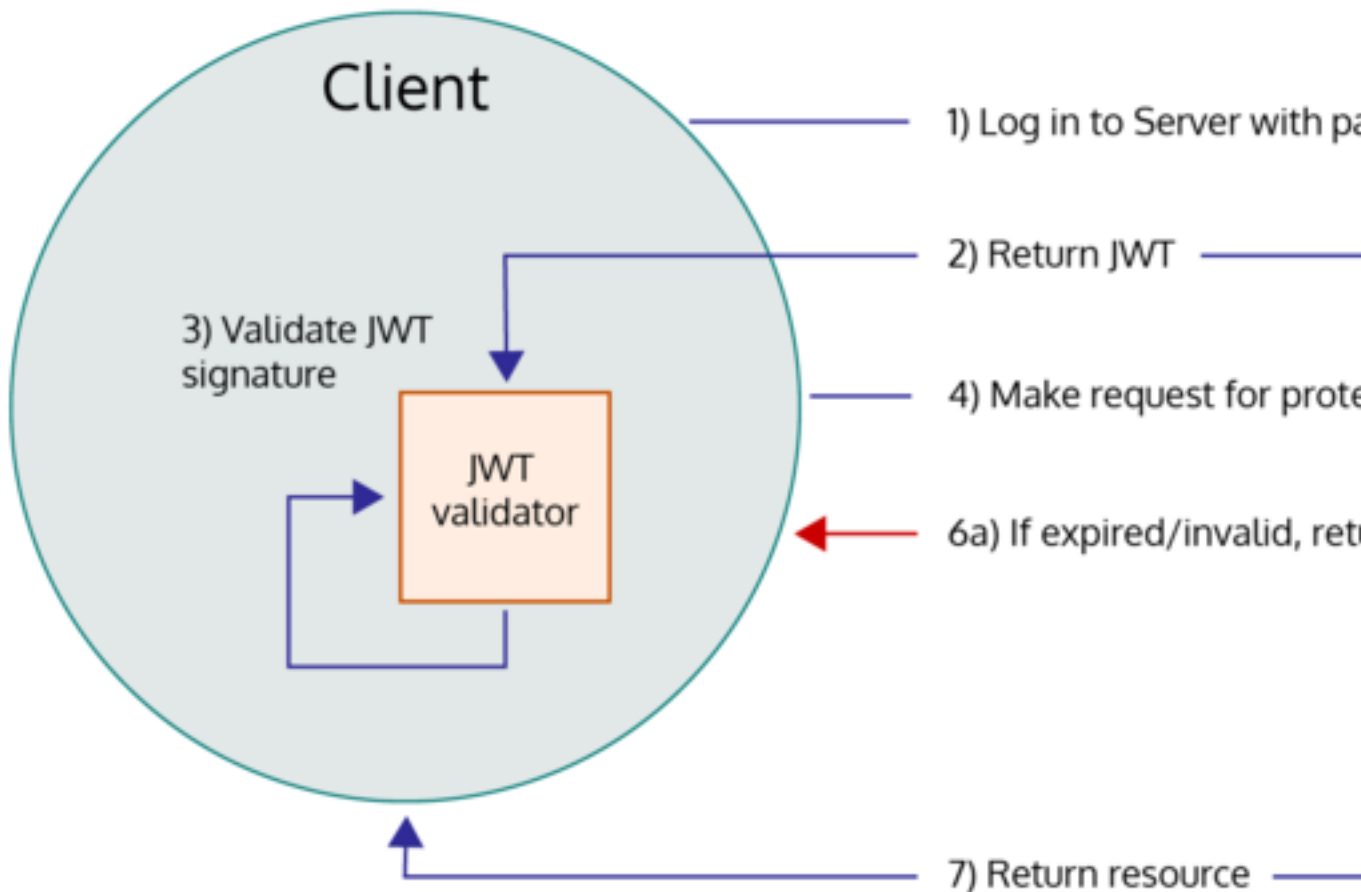
1. It uses **SQLGetConnectAttr** to check if two-factor authentication is enabled.
2. It attempts to provide the token to the server with the **SQLSetConnectAttr** call and performs error processing if this fails. If **SQLSetConnectAttr** fails, the server drops the connection, so you need to reestablish the connection before you can attempt authentication again.

```
// Given a connection called "conn"
SQLINTEGER stringLengthPtr;
SQLINTEGER attr;
SQLRETURN rc = SQLGetConnectAttr(conn, 1002, &attr, sizeof(attr), &stringLengthPtr);
if attr {
    // Prompt the user for the two-factor authentication token.
    wstring token;
    SQLRETURN rc = SQLSetConnectAttr(conn, 1002, token, SQL_NTS);
    if !rc {
        // Process the error from a invalid authentication token.
    }
}
```

JSON Web Token (JWT) Authentication

Overview of JWT Authentication

A JWT is a compact, URL-safe means of authentication, authorization, or information exchange. In the case of authentication, a server provides a JWT to an already-authenticated client so that the client does not need to reprovide a password to access protected resources on the server until the JWT expires. The authentication flow can look like the following diagram:



Configuring JWT Authentication

To use JWT authentication, InterSystems IRIS must have a configuration for issuing JWTs so that the client can validate their integrity.

- The **System Administration > Security > System Security > Authentication/Web Session Options** page includes the following settings:
 - **JWT Issuer field** — Defines the `iss` claim in the JWT payload which identifies the issuer of the JWT to the client.
 - **JWT Signature Algorithm** — Determines the encryption algorithm that InterSystems IRIS uses for signing and validating the JWT.
 - **Reset Key Store** — Rotates the encryption keys. This invalidates all previously issued, unexpired JWTs.

Issuing JWTs is not the only configuration required. You must also configure the client receiving the JWT to accept and use them. REST web applications are among the clients that can use JWTs for authentication. The [Create and Edit Applications](#) page describes these settings.

Support for JWT Usage

InterSystems IRIS supports JWT usage for authentication to a REST API and in the OAuth 2.0 framework.

With a REST API

Once configured for JWT authentication, a REST API gains four endpoints that should not be included in the `UrlMap` in the dispatch class:

- `/login` — A call to this endpoint using basic HTTP authentication or with valid credentials in the body of the request returns an access token and a refresh token that can be used in subsequent requests.
- `/logout` — A call to this endpoint, if not using Group-By-ID, invalidates the supplied access token and the associated refresh token. If using Group-By-ID, then all sessions with the current By-ID group are invalidated.
- `/refresh` — A call to this endpoint issues a new access and refresh token pair when invoked with a valid refresh token. This invalidates the previous access and refresh token pair.
- `/revoke` — If not using Group-By-ID, this is functionally the same as `/logout`. If using Group-By-ID, this revokes only the current access and refresh token pair.

You can customize the endpoint names in the dispatch class using the following:

```
Parameter TokenLoginEndpoint = "mylogin";
Parameter TokenLogoutEndpoint = "mylogout";
Parameter TokenRevokeEndpoint = "myrevoke";
Parameter TokenRefreshEndpoint = "myrefresh";
```

Accessing REST Endpoints with a JWT

You supply the access token in HTTP requests to the REST API in a header using the format of `Authorization: Bearer ACCESS_TOKEN_HERE`. Other than supplying this access token instead of your credentials in the request, you access your web application endpoints as normal except for the `/login` and `/refresh` endpoints. To retrieve the access token, you first access the `/login` endpoint.

The /login Endpoint

To access the `/login` endpoint and retrieve the access and refresh tokens, make an HTTP POST request without an authentication header and with your credentials in the body in JSON format as below:

```
{"user": "YOUR USER",  
"password": "YOUR PASSWORD"}
```

If the credentials are valid, you receive a response similar to the following:

[illegible]

Using the `/login` access token as an example, the `Authorization` header for requests to your other REST API endpoints has the value of:

You access the `/refresh` endpoint with an HTTP POST request without an access token. Instead, you send the following JSON-formatted data in the body of the request:

This returns a new access token and refresh token pair, similar to accessing the `/login` endpoint but without losing your session from a logout.

To learn more about how InterSystems IRIS supports JWTs with the OAuth 2.0 framework, see [Using OAuth 2.0 and OpenID Connect](#).

LDAP

LDAP and InterSystems IRIS®

InterSystems IRIS® provides support for authentication and authorization using LDAP, the Lightweight Directory Access Protocol. LDAP systems have a central repository of user information, from which InterSystems IRIS retrieves information. For example, on Windows, a domain controller using Active Directory is an LDAP server.

Support includes:

- [LDAP authentication](#) — InterSystems IRIS prompts users for a username and password. The instance is associated with an LDAP server, which performs authentication and retrieves the user's roles and other authorization information. The instance can also be configured to use [cached credentials](#) to authenticate users, in cases where it cannot connect to the LDAP server.
- [LDAP authorization](#) — InterSystems supports LDAP groups for specifying roles as part of [authorization](#). [LDAP authorization with OS-based authentication](#) is used for the local InterSystems IRIS terminal. (Access to the Terminal is managed by `%Service_Console` on Windows and `%Service_Terminal` on all other operating systems.)

InterSystems IRIS can also provide authentication and authorization for [multiple LDAP domains](#) simultaneously.

You can also use LDAP with the InterSystems IRIS [delegated authentication](#) feature, which allows you to implement custom mechanisms to replace the authentication and role-management activities that are part of InterSystems security.

InterSystems IRIS provides LDAP support for:

- Active Directory
- OpenLDAP
- LDAP version 3 protocols (earlier LDAP protocols are not supported)

LDAP Authentication

Overview of Setting Up LDAP Authentication

To configure an InterSystems IRIS service or application to use an LDAP server for authentication:

1. Configure InterSystems IRIS to use the LDAP server:
 - a. [Enable LDAP and related features](#) for the instance.
 - b. [Create an LDAP configuration](#) for the instance of InterSystems IRIS. This includes specifying the names of LDAP user properties to be used for setting the values of properties of InterSystems IRIS users.
 - c. Optionally, [test the LDAP configuration](#).
 - d. Optionally, configure the instance to support [multiple LDAP domains](#).
 - e. [Set up a role that is required for logging in to the instance](#).
 - f. [Enable LDAP for the instance's relevant services and applications](#). This involves enabling LDAP for the entire instance of InterSystems IRIS and then enabling it for the relevant services or applications.

Note: To perform LDAP authentication programmatically, use InterSystems IRIS [delegated authentication](#).

Enable LDAP for an Instance

The first step in configuring an instance of InterSystems IRIS to use LDAP is to enable the features you wish to use:

1. From the Management Portal home page, go to the **Authentication/Web Session Options** page (**System Administration > Security > System Security > Authentication/Web Session Options**).
2. On the **Authentication/Web Session Options** page:
 - To enable LDAP authentication, select **Allow LDAP authentication**.
 - To enable authentication using LDAP cached credentials, select **Allow LDAP cache credentials authentication**. For more information on this topic, see [LDAP Cached Credentials](#).
3. Click **Save** to apply the changes.

LDAP Cached Credentials

If you configure an instance to use *LDAP cached credentials*, it stores (caches) a copy of the credentials that it most recently used to authenticate each user. If an instance supports cached credentials and it cannot connect to the LDAP server, then it uses the cached LDAP credentials to authenticate users. This can be caused by an issue with the LDAP server itself or with the connection to the server.

To secure cached credentials, InterSystems IRIS stores all LDAP passwords in the security database as a one-way hash. If the instance cannot use the LDAP server to validate the user, it then attempts to confirm that:

- The hash of the entered password matches the hash of the stored password
- The cached expiration date from the last LDAP login has not been reached

If both conditions are true, the instance authenticates the user and login proceeds; otherwise, login fails.

Create or Modify an LDAP Configuration

To perform LDAP authentication, InterSystems IRIS uses an *LDAP configuration*. An LDAP configuration specifies a connection to an LDAP server for a particular security domain and has information required to:

- Connect to and query the LDAP server
- Retrieve the required information about the user being authenticated

Note: If Kerberos is enabled for an instance, all menu items and other labels for LDAP configurations refer to LDAP/Kerberos configurations. The following procedure does not note this in each individual situation.

To create or modify an LDAP configuration:

1. Go to the Management Portal **Security LDAP Configurations** page (**System Administration > Security > System Security > LDAP Configurations**).

During installation, if you are installing InterSystems IRIS onto a machine that is currently using an LDAP server, InterSystems IRIS creates an LDAP configuration based on that LDAP server's domain and other configuration information.

2. Create or modify a configuration:
 - To modify an existing configuration, click its name. For example, if you are using the configuration associated with the local LDAP server, then you may simply wish to check this configuration's attributes and modify any as needed.
 - To create a configuration, click the **Create New LDAP Configuration** button. This displays the **Edit LDAP configuration** page.

Note: When creating a configuration, on the **Edit LDAP configuration** page, select the **LDAP configuration** check box if it is available. This displays the fields that define the LDAP configuration.

3. Modify or complete the fields to define the configuration (listed [below](#)).
4. If you create multiple configurations, you must specify which one is the default on the **System-wide Security Parameters** page (**Security Administration > Security > System Security > System-wide Security Parameters**), using the **Default security domain** drop-down.

LDAP Configuration Fields

An LDAP configuration includes the following fields:

- **Login Domain Name** — *Required*. The name of the LDAP configuration. This is typically in the form of example.com or example.org.

If you enter a value that does not include a period, the system appends .com to it, so that example becomes example.com. If you enter a value in uppercase, the system puts in lowercase, so that EXAMPLE.COM becomes example.com. The system performs both transformations, if appropriate.

The system uses the transformed value of the **Name** field to populate the **LDAP Base DN to use for searches** field.

- **Description** — Any text to describe the configuration.
- **Copy from** — *Available only when creating a configuration*. Whether or not InterSystems IRIS copies attributes from an existing LDAP configuration to specify initial values for this one.
- **LDAP Enabled** — Whether or not InterSystems IRIS can use the configuration to connect to an LDAP server.
- **LDAP server is a Windows Active Directory server** — *Windows only*. Whether or not the LDAP server is a Windows Active Directory server.
- **LDAP hostnames** — *Required*. The name(s) of the host(s) on which the LDAP server is running. The complexity of each hostname can range from an unqualified hostname to fully-qualified hostname with a port number; the required form of the hostname(s) depends on the particular configuration.

If the LDAP server is configured to use a particular port, you can specify it by appending “:portname” to the hostname; typical usage is not to specify a port and to let the LDAP functions use the default port. You can specify the domain example.com as your hostname if you have multiple replicated domain servers on your network like:

```
ldapservers.example.com  
ldapservers1.example.com  
ldapservers2.example.com  
ldapservers3.example.com
```

LDAP performs a DNS query for the addresses of all the matching LDAP servers and then automatically selects one to connect to.

Important: Including a port number in the value of **LDAP hostnames** affects the TLS behavior when establishing a connection:

- If the value specified contains a port number *other than 636*, such as `ldapservers.example.com:389` and the **Use TLS/SSL encryption for LDAP sessions** check box is selected, then the instance attempts to establish a plaintext connection to the LDAP server and then issue a StartTLS command to encrypt the connection.
- If the value specified for LDAP hostnames contains the port number *636*, such as `ldapservers.example.com:636`, then the instance attempts to establish a TLS connection with the LDAP server directly—whether or not the **Use TLS/SSL encryption for LDAP sessions** check box is selected. Note, however, that connecting directly to port 636 from UNIX® client instances is not supported.

For background, see the class reference for the `%SYS.LDAP.Init()` method.

- LDAP search information — varies by circumstances:

- **LDAP username to use for searches** — *For Windows Active Directory servers only. Required if available.* The user name provided to the LDAP server to establish an initial connection and which is used to perform LDAP searches and lookups. This user is also known as the *search user*.

The search user must have permission to read the entire LDAP database. It is important to ensure that the search user has uninterrupted access to the LDAP database. For example, the user's LDAP account should be set so that:

- The user cannot change the account's password
- The password never expires
- The account never expires

For more information on searching the LDAP database, see [How LDAP Looks Up the Target User in Its Database](#).

- **LDAP search user DN** — *For all non-Windows platforms and Windows non-Active Directory servers. Required if available.* The Distinguished Name (DN) of the user provided to the LDAP server to establish an initial connection and which is used to perform LDAP searches and lookups. This user is also known as the *search user*.

The search user must have permission to read the entire LDAP database. It is also important to ensure that the search user has uninterrupted access to the LDAP database. For example, the user's LDAP account should be set so that:

- The user cannot change the account's password
- The password never expires
- The account never expires

For example, if the search user is "ldapsearchuser", the LDAP DN (distinguished name) might be as follows:

```
uid=ldapsearchuser,ou=People,dc=example,dc=com
```

For more information on searching the LDAP database, see [How LDAP Looks Up the Target User in Its Database](#).

- **LDAP username password** — *Available only when creating or modifying a configuration.* The password associated with the account used for the initial connection.
- **LDAP Base DN to use for searches** — *Required.* The point in the directory tree from which searches begin. This typically consists of domain components, such as `DC=example,DC=com`.
- **LDAP Base DN for Groups to use for searches** — *Required.* The point in the directory tree from which searches for [nested groups](#) begin. This typically consists of organizational units and domain components, such as `OU=IRIS,OU=Groups,DC=test,DC=com`. By default, this is set to the same value as **LDAP Base DN to use for searches**.
- **LDAP Unique search attribute** — *Required.* A unique identifying element of each record, which therefore makes it appropriate for searches. For more information on searching the LDAP database, see [How LDAP Looks Up the Target User in Its Database](#).
- **Use TLS/SSL encryption for LDAP sessions** — Whether or not the InterSystems IRIS instance and the LDAP server encrypt their communications using TLS (disabled by default).

Important: InterSystems recommends that you enable TLS encryption for LDAP.

For connections to Active Directory servers, note the following:

- When enabled for an LDAP connection from an instance on Windows to an Active Directory server, the connection uses port 636 (which is a TLS-encrypted port).

- When enabled for an LDAP connection from an instance on UNIX® to an Active Directory server, InterSystems IRIS first establishes the connection on port 389 (the unencrypted LDAP port); encryption is then turned on by a **StartTLS** call.

InterSystems also recommends setting the *LDAP server signing requirements* parameter to `Require signature` on the Active Directory Server. This prevents any LDAP **bind** command on the server on port 389 to be executed unless the channel is encrypted with StartTLS. For more information, see [Domain Controller: LDAP Server Signing Requirements](#) article on the Microsoft web site.

- **File with Certificate Authority certificate(s) to authenticate the LDAP server** — *UNIX® only*. The location of the file containing any TLS certificates (in PEM format) being used to authenticate the server.

On Windows, to specify the location of a file containing any TLS certificates (in PEM format) being used to authenticate the server certificate to establish a secure LDAP connection, use [Microsoft Certificate Services](#). Certificates must be installed in the Certificates (Local Computer)\Trusted Root Certification Authorities certificate store.

- **Allow ISC_LDAP_CONFIGURATION environment variable** — If you are using [OS-based LDAP](#) and [multiple domains](#), specifies whether or not to use the *ISC_LDAP_CONFIGURATION* environment variable. If the environment variable is defined, then OS-based LDAP uses it to determine which LDAP configuration to use for authentication.
- **Use LDAP Groups for Roles/Routine/Namespace** — Whether or not the user's roles, routine, and namespace come from the user's group memberships (true by default); if not, then they come from the attribute fields of the user's LDAP record. If you select this field, the system enables and disables other fields (see each subsequent field for details).

Note: InterSystems recommends the use of LDAP groups for authorization, rather than LDAP attributes (including InterSystems registered LDAP properties). If you have existing code or are otherwise required to use registered properties, see [Configure Authorization with LDAP Attributes](#) for details.

- **Search Nested Groups for Roles/Routine/Namespace** — *Only active if LDAP server is a Windows Active Directory server and Use LDAP Groups for Roles/Routine/Namespace are selected.* Whether or not search returns all of a user's nested groups. See [Nested Groups](#) for more information on nested groups.
- **Organization ID prefix for group names** — *Only active if Use LDAP Groups for Roles/Routine/Namespace is selected.* See [LDAP Group Name Configuration](#) for more information.
- **Allow Universal group Authorization** — *Only active if Use LDAP Groups for Roles/Routine/Namespace is selected.* Whether or not searches use the attributes on the LDAP server that are relevant for all InterSystems IRIS instances. See [Create Universal LDAP Authorization Groups](#) for more information.
- **Authorization Group ID** — *Only active if Use LDAP Groups for Roles/Routine/Namespace is selected.* The multiple-instance group to which this instance belongs. See [Create LDAP Authorization Groups for Multiple Instances \(Multiple-Instance Groups\)](#) for more information.
- **Authorization Instance ID** — *Only active if Use LDAP Groups for Roles/Routine/Namespace is selected.* The single-instance group to which this instance belongs. See [Create LDAP Authorization Groups for a Single Instance \(Single-Instance Groups\)](#) for more information.
- **User attribute to retrieve default namespace** (not active if LDAP groups are selected) — The attribute whose value is the source for the *Startup namespace* property for a user. This property of an InterSystems IRIS user is described in [User Account Properties](#); this LDAP property is described in [Configure Authorization with LDAP Attributes](#).
- **User attribute to retrieve default routine** (not active if LDAP groups are selected) — The attribute whose value is the source for the *Tag^Routine* property for a user. This property of an InterSystems IRIS user is described in [User Account Properties](#); this LDAP property is described in [Configure Authorization with LDAP Attributes](#).
- **User attribute to retrieve roles** (not active if LDAP groups are selected) — The attribute whose value determines the roles to which a user is assigned. When creating this attribute, it must be specified as an LDAP multivalued attribute.

For information about an InterSystems IRIS user's [roles](#), see the **Roles** tab of a user's **Edit User** page; this LDAP property is described in [Configure Authorization with LDAP Attributes](#).

- **User attribute to retrieve comment attribute** — The attribute whose value is the source for the *Comment* property for a user. This property is described in [User Account Properties](#). Once a user has logged in, you can retrieve the value of this property using the `Security.Users.Get()` method.
- **User attribute to retrieve full name from** — The attribute whose value is the source for the *Full name* property for a user. This property is described in [User Account Properties](#). Once a user has logged in, you can retrieve the value of this property using the `Security.Users.Get()` method.
- **User attribute to retrieve mail address** — The attribute whose value is the source for the *Email address* property for a user. This property is described in [User Account Properties](#). Once a user has logged in, you can retrieve the value of this property using the `Security.Users.Get()` method.
- **User attribute to retrieve mobile phone** — The attribute whose value is the source for the *Mobile Phone Number* property for a user. This property is described in [User Account Properties](#). Once a user has logged in, you can retrieve the value of this property using the `Security.Users.Get()` method.
- **User attribute to retrieve mobile provider from** — The attribute whose value is the source for the *Mobile Phone Service Provider* property for a user. This property is described in [User Account Properties](#). Once a user has logged in, you can retrieve the value of this property using the `Security.Users.Get()` method.
- **LDAP attributes to retrieve for each user** — Any attributes whose values are the source for any application-specific variables. Application code can then use the `Get` method of the `Security.Users` class to return this information.

The values of the fields of an LDAP configuration are stored in an instance of the `Security.LDAPConfigs` class.

Note on LDAP/Kerberos Configuration Fields

If Kerberos authentication is enabled for an instance, then the page for creating an LDAP configuration is **Edit LDAP/Kerberos configurations** page. It has the same fields as the **Edit LDAP configurations** page, as described in [LDAP Configuration Fields](#).

Test an LDAP Configuration

Once you have created an LDAP configuration, you can test it. This allows you to confirm that it properly connects to the LDAP server or troubleshoot any issues that arise. To test a configuration:

1. In the Management Portal, go to the **Security LDAP Configurations** page (**System Administration > Security > System Security > LDAP Configurations**).
2. Click **Test LDAP Authentication**.
3. In the **Username** and **Password** fields, enter a valid username and password defined on the LDAP server. If the instance is configured to use multiple domains, you must provide a fully qualified username, such as `EndUser@example.com`; if the instance is using only a single domain, simply enter the unqualified username (without the `@` symbol or the domain name), such as `EndUser`.
4. Click **Test**.

The **Test Results** field displays output from the LDAP server.

Note: This feature only tests if an instance can connect to an LDAP server and perform authentication checks for the entered user. It does not perform any authorization or permission checks to determine if the user can successfully log in to the system.

If the test succeeds for the entered user, but the user cannot log in, then check the audit record for the login failure. To ensure successful login, you may need to give additional permissions to the user.

Use Multiple LDAP Domains

InterSystems IRIS supports LDAP authentication with multiple domains. This allows the instance to have user accounts that include the same username from more than one domain, such as `EndUser@example.com` and `EndUser@otherexample.com`. This feature can be useful in multiple scenarios. For example:

- It allows merging distinct sets of users from multiple domains into one larger group while preserving unique identifiers for each user.
- It allows the same individual to have accounts on multiple domains with varying privileges for each.

To use multiple domains:

1. Create additional LDAP configurations according to the instructions in [Create or Modify an LDAP Configuration](#).
2. Configure the instance to use multiple domains and then specify a default domain:
 - a. Enable the use of multiple domains for the instance. In the Management Portal, on the **System-wide Security Parameters** page (**System Administration** > **Security** > **System Security** > **System-wide Security Parameters**), select the **Allow multiple security domains** check box.
 - b. Specify a default domain. In the Management Portal, on the **System-wide Security Parameters** page (**System Administration** > **Security** > **System Security** > **System-wide Security Parameters**), select a default domain using the **Default security domain** drop-down.
 - c. Click **Save**.

For more information about this page, see [System-Wide Security Parameters](#).

Note: Even if you are using multiple domains, the name for each user must be unique, even if they are of different types. Hence, if you create a user such as `EndUser@example.com` that is a password user, you cannot then log in to InterSystems IRIS through LDAP as the user `EndUser@example.com`, as InterSystems IRIS cannot create the account for `EndUser@example.com` as an LDAP user.

Set Up a Required Login Role

If you have multiple instances of InterSystems IRIS and are using LDAP authentication or [OS-based authentication with LDAP authorization](#), then InterSystems *strongly* recommends that each instance have a role that is required for the users who are connecting to it. This mechanism prevents users from accessing instances where they are insufficiently privileged; otherwise, a user who holds various roles on one instance may then have those same roles on an instance where this is not intended.

To set up a required login role:

1. For each instance, if the role to be required does not already exist, create it according to the instructions in [Create Roles](#).
2. For each instance, specify the required role in the **Role required to connect to this system** field on the **System Security Settings** page (**System Administration** > **Security** > **System Security** > **System-wide Security Parameters**).
3. Add an LDAP group with a name that includes the name of the required role. The name of the group is of the form:

`intersystems-Instance-instanceID-Role-rolename`

where:

- *instanceID* is the unique identifier for the instance on the LDAP server
- *rolename* is the name of the role required to connect

Note: In certain circumstances, such as with mirroring, you may prefer to have a single required login role among multiple instances.

For example, suppose there are two systems, *TEST* and *PRODUCTION*. To secure each of these systems, create a role on **TEST** called **TESTACCESS** and a role on **PRODUCTION** called **PRODUCTIONACCESS**. On **TEST**, set the value of the **Role required to connect to this system** field to **TESTACCESS**; on **PRODUCTION**, set it to **PRODUCTIONACCESS**. Then, if a user is only allowed to access the **TEST** system, assign that user the **TESTACCESS** role *only* and do not assign the **PRODUCTIONACCESS** role to the user. For users who can access either system, assign them both **PRODUCTIONACCESS** and **TESTACCESS** roles.

Enable LDAP for Services and Applications

After enabling LDAP authentication for the instance, enable it for the instance's relevant services or applications:

1. Because LDAP authentication is enabled for the instance, an **LDAP** check box appears on the **Edit Service** page for the services that support LDAP authentication and the **Edit Web Application** page for web applications.
2. Enable LDAP authentication for services and applications as appropriate.

The following services support LDAP authentication:

- `%Service_Bindings`
- `%Service_CallIn`
- `%Service_ComPort`
- `%Service_Console`
- `%Service_Login`
- `%Service_Terminal`
- `%Service_Telnet`
- `%Service_WebGateway`

These fall into several categories of access modes:

- **Local Access** —
`%Service_CallIn, %Service_ComPort, %Service_Console, %Service_Login, %Service_Terminal, %Service_Telnet`
 To use LDAP authentication with local connections, enable it for the service.
- **Client-Server Access** —
`%Service_Bindings`
 To use LDAP authentication with client-server connections, enable it for the service.
- **Web Access** —
`%Service_WebGateway`

To allow named users to log in to web applications using LDAP authentication, you will have to enable the relevant web applications to use LDAP. See [Security Settings](#) in “[Defining Applications](#)” for more information about adding authentication mechanisms to a web application. Enabling LDAP authentication for the service also allows the Web Gateway itself to authenticate using LDAP authentication.

The State of an Instance After LDAP Authentication

Any user who is initially authenticated using LDAP authentication is listed in the table of users on the **Users** page (**System Administration > Security > Users**) as having a Type of “LDAP user”. If a system administrator has explicitly created a user through the Management Portal (or using any other native InterSystems IRIS facility), that user has a type of “InterSystems IRIS password user”. If a user attempts to log in using LDAP authentication and is successfully authenticated, InterSystems IRIS determines that this user already exists as an InterSystems IRIS user — not an LDAP user — and so login fails.

View an LDAP Configuration in the Portal As %Operator

If you are logged in to the Management Portal as a user who has the `%Operator` role or the `%Admin_Operate:Use` privilege, you can view (but not edit) the instance’s LDAP configurations:

1. In the Portal, go to the **LDAP Configurations** page (**System Operation > LDAP Configurations**).
2. On that page, click on the name of the configuration you wish to view, which displays the **Display LDAP Configuration** for that configuration.

To edit an LDAP configuration, go to the **Security LDAP Configurations** page (**System Administration > Security > System Security > LDAP Configurations**); you must have the `%Admin_Secure:Use` privilege.

The Security LDAP Configurations Page

The Portal’s **Security LDAP Configurations** page (**System Operation > LDAP Configurations**) displays a list of the instance’s LDAP configurations. Click the name of a configuration to view its [properties](#). If Kerberos authentication is enabled for the instance, this is called the **Security LDAP/Kerberos configurations** page (**System Operation > LDAP/Kerberos configurations**).

LDAP Authorization

In addition to performing authentication with LDAP, InterSystems IRIS supports LDAP authorization. InterSystems recommends the use of LDAP groups rather than LDAP attributes for managing role, routine, and namespace definitions.

Overview of Configuring LDAP Authorization

To configure an InterSystems service or application to use LDAP for authorization:

1. Configure the instance for LDAP or OS-based authentication
2. For LDAP authorization:
 - a. Design the groups for [LDAP authorization on InterSystems IRIS instances](#)
 - b. [Configure the LDAP server](#) to use those groups

Configure Authorization with LDAP Groups

- [LDAP Groups and InterSystems IRIS](#)
- [LDAP Authorization Group Models](#):
 - [Create LDAP Authorization Groups for a Single Instance \(Single-Instance Groups\)](#)
 - [Create LDAP Authorization Groups for Multiple Instances \(Multiple-Instance Groups\)](#), including [Authorization for Mirroring](#)
 - [Create Universal LDAP Authorization Groups](#)
- [Other Topics for LDAP Authorization with LDAP Groups](#)

LDAP Groups and InterSystems IRIS

LDAP groups allow you to assign privileges to users using an LDAP server:

- The schema on the LDAP server specifies the names of groups. Typically, the LDAP administrator defines these names; InterSystems IRIS uses one of three predefined name structures described below.
- Each group has a distinguished name (DN) that uniquely identifies it.
- Each group specifies access to an InterSystems IRIS role, routine, or namespace

InterSystems IRIS supports LDAP groups that provide authorization for:

- A single instance
- Multiple instances
- All instances

To set up groups for InterSystems IRIS:

1. Determine if you are going to use groups for a single instance, for multiple instances, or for all instances.
2. Create one or more groups with names that follow the appropriate naming convention. Each group specifies a user's role, default namespace, or default routine; since a user can have multiple roles, it is valid to belong to multiple groups that specify roles.

Note: Note that when defining these groups on your LDAP server, they should be created as *security* groups, and not *distribution* groups.

3. Configure your LDAP users to specify which ones belong to which groups. This requires that, for each user's LDAP account, you assign the user to multiple groups to specify one or more roles, a default namespace, and a default routine. This determines which roles each user has after logging in, the user's default namespace, and the user's default routine.
4. Configure the local InterSystems IRIS instance so that there are definitions for all the roles that are specified on the LDAP server.

LDAP Authorization Group Models

InterSystems IRIS supports for three kinds of group authorization using LDAP.

- [Create LDAP authorization groups for a single instance \(single-instance groups\)](#)
- [Create LDAP authorization groups for multiple instances \(multiple-instance groups\)](#), including [mirroring](#)
- [Create universal LDAP authorization groups](#)

Create LDAP Authorization Groups for a Single Instance (Single-Instance Groups)

InterSystems IRIS allows you to create LDAP groups that provide authorization for only a single instance; hence, each of these is known as a *single-instance group*. To create this kind of authorization group:

1. On the InterSystems IRIS instance, confirm or modify the value of the LDAP parameter **Authorization Instance ID**. By default, its value is *NodeName_InstanceName*, where *NodeName* is the machine on which the InterSystems IRIS instance is running and *InstanceName* is the name of that instance.

To set the parameter's value manually:

- a. In the Management Portal, go to the **Security LDAP Configurations** page (**Management Portal > System Administration > Security > System Security > LDAP Configurations**).
- b. On that page, select the configuration to edit by clicking on its name.

- c. On the page for editing the configuration that appears, select **Use LDAP Groups for Roles/Routine/Namespaces**.
 - d. Next, in the **Authorization Instance ID** field, enter the value for the parameter and click **Save**.
2. On the LDAP server, define role, namespace, and routine groups with names that conform to the required InterSystems structure and that use the *Instance* keyword, followed by the value of the **Authorization Instance ID**. Note that these strings are not case sensitive. These group names are of the form:

`intersystems-Instance-AuthorizationInstanceIDValue-Role-RoleName`

`intersystems-Instance-AuthorizationInstanceIDValue-Routine-RoutineName`

`intersystems-Instance-AuthorizationInstanceIDValue-Namespace-NamespaceName`

where:
 - *AuthorizationInstanceIDValue* is the value specified for the **Authorization Instance ID** field
 - *RoleName*, *RoutineName*, and *NamespaceName* are each the name of the role, default routine, or default namespace.

Note: A user can have any number of roles; typically, access to the system requires at least one role. A user can have only one default routine and one default namespace; however, these are not required, so a user may have no default routine and no default namespace.

 - *RoleName* can include multiple roles, delimited by a “^”. For example, “%All^Admin^Application4” includes the “%All”, “Admin”, and “Application4” roles.
3. On the InterSystems IRIS instance, configure a role associated with each group.

For example, suppose you are running an application on an instance called *Test* that is on a machine called *Node1*. You wish to set up three categories of users:

- Application users — Can only run the application
- Administrative users — Can run various administrative tools and the application
- Superusers — Have full access

To set up this authorization model, create the following groups on the LDAP server:

```
intersystems-Instance-Node1_Test-Role-Administrator
intersystems-Instance-Node1_Test-Role-LocalApplication
intersystems-Instance-Node1_Test-Role-%All
intersystems-Instance-Node1_Test-Routine-LocalApplication
intersystems-Instance-Node1_Test-Routine-%SS
intersystems-Instance-Node1_Test-Routine-%prmode
intersystems-Instance-Node1_Test-Namespace-%SYS
intersystems-Instance-Node1_Test-Namespace-USER
```

Next, create the roles that corresponds to each category of user:

- Administrator
- LocalApplication

Note: You do not need to create a **%All** role, because it already exists.

Finally, create the three categories of users:

- Application users — Can run only the application, *LocalApplication*; are assigned to the following LDAP groups:

- intersystems-Instance-Node1_Test-Role-LocalApplication
 - intersystems-Instance-Node1_Test-Routine-LocalApplication
 - intersystems-Instance-Node1_Test-Namespace-USER
- Administrative users — Can run various administrative tools and the application; are assigned to the following LDAP groups:
 - intersystems-Instance-Node1_Test-Role-LocalApplication
 - intersystems-Instance-Node1_Test1-Role-Administrator
 - intersystems-Instance-Node1_Test-Routine-%SS
 - intersystems-Instance-Node1_Test-Namepace-%SYS
 - Superusers — Have **%All** access; are assigned to the following LDAP groups:
 - intersystems-Instance-Node1_Test-Role-%All
 - intersystems-Instance-Node1_Test-Namepace-%SYS
 - intersystems-Instance-Node1_Test-Routine-%pmode

Create LDAP Authorization Groups for Multiple Instances (Multiple-Instance Groups)

InterSystems IRIS allows you to create LDAP groups that provide authorization for multiple instances; hence, each of these is known as a *multiple-instance group*. To create this kind of authorization group:

1. Determine how the various instances are sharing information among groups. This determines the group for each instance and the information to which users have access.
2. For each instance in the group, modify the value of the LDAP parameter **Authorization Group ID** to be the same as the other instances in the group.

To set the parameter's value manually:

- a. In the Management Portal, go to the **Security LDAP Configurations** page (**Management Portal > System Administration > Security > System Security > LDAP Configurations**).
 - b. On that page, select the configuration to edit by clicking on its name.
 - c. On the page for editing the configuration that appears, select **Use LDAP Groups for Roles/Routine/Namepace**.
 - d. Next, in the **Authorization Group ID** field, enter the value for the parameter and click **Save**.
3. On the LDAP server, set up role, namespace, and routine groups that conform to the required InterSystems structure and that use the **Group** keyword, followed by the value of the **Authorization Group ID**. Note that these strings are not case sensitive. These group names are of the form:

intersystems-Group-AuthorizationGroupIDValue-Role-RoleName

intersystems-Group-AuthorizationGroupIDValue-Routine-RoutineName

intersystems-Group-AuthorizationGroupIDValue-Namespace-NamepaceName

where:

- *AuthorizationGroupIDValue* is the value specified for the Authorization Group ID field
- *RoleName*, *RoutineName*, and *NamespaceName* are each the name of the role, default routine, or default namespace.

Note: A user can have any number of roles; typically, access to the system requires at least one role. A user can have only one default routine and one default namespace; however, these are not required, so a user may have no default routine and no default namespace.

- *RoleName* can include multiple roles, delimited by a “^”. For example, “%All^Admin^Application4” includes the “%All”, “Admin”, and “Application4” roles.

4. Configure the required roles on all the instances that are using them.

For example, suppose you have seven ECP application servers attached to five database servers. Two of the database servers are a failover pair, and the other three are async reporting members. All these servers (both the application servers and the database servers) run the SALES application. The application’s end users need a more limited set of privileges and its administrative users need greater privileges. Hence, you set up three categories of users:

- Application users — Can only run the application
- Application server administrators — Can run the application; have full access to the application servers and no access to the database servers
- Database administrators — Have full access to the application servers and administrative access to the database servers

To configure LDAP authorization to support these requirements:

- Set the **Authorization Group ID** on the applications servers to SALESAPP
- Set the **Authorization Group ID** on the database servers to SALESDB

On the LDAP server, define the groups as follows:

```
intersystems-Group-SALESAPP-Role-%All
intersystems-Group-SALESAPP-Role-LocalApplication
intersystems-Group-SALESAPP-Routine-LocalApplication
intersystems-Group-SALESAPP-Routine-%pmode
intersystems-Group-SALESAPP-Namespace-USER
intersystems-Group-SALESAPP-Namespace-%SYS
intersystems-Group-SALESDB-Role-Administrator
intersystems-Group-SALESDB-Routine-INTEGRIT
intersystems-Group-SALESDB-Namespace-%SYS
```

Next, create the roles that corresponds to each category of user:

- Administrator
- LocalApplication

Note: You do not need to create a %All role, because it already exists.

Finally, create the three categories of users:

- Application users – Can only run the application, LocalApplication; are assigned to the following LDAP groups:
 - intersystems-Group-SALESAPP-Role-LocalApplication
 - intersystems-Group-SALESAPP-Routine-LocalApplication
 - intersystems-Group-SALESAPP-Namespace-USER
- Application server administrators — Can run the application, have full access to the application servers, and have no access to the database servers; are assigned to the following LDAP groups:

- intersystems-Group-SALESAPP-Role-LocalApplication
 - intersystems-Group-SALESAPP-Namespace-USER
 - intersystems-Group-SALESAPP-Role-%All
 - intersystems-Group-SALESAPP-Routine-%pmode
- Database administrators — Have full access to the application servers and administrative access to the database servers; are assigned to the following LDAP groups:
 - intersystems-Group-SALESAPP-Role-%All
 - intersystems-Group-SALESAPP-Routine-%pmode
 - intersystems-Group-SALESAPP-Namespace-%SYS
 - intersystems-Group-SALESDB-Role-Administrator
 - intersystems-Group-SALESDB-Routine-INTEGRIT
 - intersystems-Group-SALESDB-Namespace-%SYS

At this point, there is a fully functioning authorization model, but it does not include any superuser access to the database servers (that is, with %All). To add such access, create and add users to the following new group:

```
intersystems-Group-SALESDB-Role-%All
```

Configure LDAP Authorization Groups with Mirroring

In you are using LDAP and mirroring, InterSystems recommends using multiple-instance LDAP groups to configure authorization. Create the required multiple-instance groups and configure all the users on all members (including any async members) to use these groups.

Consider the following example, which is based on the group structure defined in the example above. Suppose, additionally:

- There is a mirror called SALESDBMIR which is a failover pair and three reporting async members
- You wish to have users with %All, but only on the failover pair

To configure authorization for this mirror:

1. To provide full access to the failover pair, create the group


```
intersystems-Group-SALESDBMIRFAILOVER-Role-%All
```
2. To provide full access to the asynchronous members, create the group


```
intersystems-Group-SALESDBMIRASYNC-Role-%All
```
3. Set the LDAP parameter **Authorization Instance ID** on each member in the failover pair to SALESDBMIR-FAILOVER.

Important: Because a disaster recovery (DR) async member may be promoted to failover member, the **Authorization Instance ID** for any DR async should also be set to SALESDBMIRFAILOVER

4. Set the LDAP parameter **Authorization Group ID** on the mirror's asynchronous members to SALESDBMI-RASYNC.
5. Next, create the mirror administrators, who have %All access to the application servers; administrative access to the nonmirrored database servers; and %All access to the failover pair only. These users are assigned to the following LDAP groups:

- intersystems-Group-SALESAPP-Role-%All
 - intersystems-Group-SALESAPP-Routine-%pmode
 - intersystems-Group-SALESAPP-Namespace-%SYS
 - intersystems-Group-SALESDB-Role-Administrator
 - intersystems-Group-SALESDB-Routine-INTEGRIT
 - intersystems-Group-SALESDB-Namespace-%SYS
 - intersystems-Group-SALESDBMIRFAILOVER-Role-%All
6. Finally, create the full administrators, who have **%All** access to all the members (the application servers, the database servers, the failover pair, and the asynchronous members). These users are assigned to the following LDAP groups:
- intersystems-Group-SALESAPP-Role-%All
 - intersystems-Group-SALESDB-Role-%All
 - intersystems-Group-SALESDBMIRFAILOVER-Role-%All
 - intersystems-Group-SALESDBMIRASYNC-Role-%All

Create Universal LDAP Authorization Groups

InterSystems IRIS allows you to create LDAP groups that provide authorization for all its instances that use a single LDAP server; these are known as *universal authorization groups*. To create this kind of authorization group:

1. Enable the use of universal authorization groups for the current instance:
 - a. In the Management Portal, go to the **Security LDAP Configurations** page (**Management Portal > System Administration > Security > System Security > LDAP Configurations**).
 - b. On that page, select the configuration to edit by clicking on its name, which displays the page for editing that configuration.
 - c. On the page for editing the configuration, select **Use LDAP Groups for Roles/Routine/Namespace**.
 - d. Select **Allow Universal group Authorization**.
 - e. Click **Save**.
2. On the LDAP server, set up role, namespace, and routine groups that conform to the required InterSystems structure. Note that these strings are not case sensitive. These group names are of the form:

intersystems-Role-*RoleName*

intersystems-Routine-*RoutineName*

intersystems-Namespace-*NamespaceName*

where *RoleName*, *RoutineName*, and *NamespaceName* are each the name of the role, default routine, or default namespace. *RoleName* can include multiple roles, delimited by a “^”. For example, “%All^Admin^Application4” includes the “%All”, “Admin”, and “Application4” roles.

Note: A user can have any number of roles; typically, access to the system requires at least one role. A user can have only one default routine and one default namespace; however, these are not required, so a user may have no default routine and no default namespace.

3. Configure the required roles on all the instances that are using the LDAP server.

For example, suppose you have an application called LocalApplication and you wish to grant various levels of access to it for users on all the InterSystems IRIS instances that use your LDAP server. Define the following LDAP groups:

```
intersystems-Role-%All
intersystems-Role-Administrator
intersystems-Role-LocalApplication
intersystems-Routine-%SS
intersystems-Routine-LocalApplication
intersystems-Namespace-USER
intersystems-Namespace-%SYS
```

Next, create the roles that corresponds to each category of user:

- Admin
- LocalApplication

Note: You do not need to create a %All role, because it already exists.

Finally, create the three categories of users:

- Application users – Have access to the application on all servers; are assigned to the following LDAP groups:
 - intersystems-Role-LocalApplication
 - intersystems-Routine-LocalApplication
 - intersystems-Namespace-USER
- Administrators — Have administrative access to all servers; are assigned to the following LDAP groups:
 - intersystems-Role-Administrator
 - intersystems-Routine-%SS
 - intersystems-Namespace-%SYS
- Superusers — Have full access to all servers; are assigned to the following LDAP groups:
 - intersystems-Role-%All

Other Topics for LDAP Authorization with LDAP Groups

Topics include:

- [LDAP Group Definition Structure](#)
- [LDAP Group Name Configuration](#)
- [Mix Different Kinds of Groups](#)
- [Nested Groups](#)
- [How LDAP Groups Regulate Access to InterSystems IRIS](#)

LDAP Group Definition Structure

Group definitions typically include:

- The group name
- A declaration of the group's organizational unit: OU=Groups
- A declaration of the domain component (DC) such as DC=example, DC=com

- Any other required information

For example, some possible group definitions might be:

```
CN=intersystems-Role-Administrator,OU=Groups,DC=intersystems,DC=com
CN=intersystems-Group-MyGroup-Namespace-USER,OU=Groups,DC=intersystems,DC=com
CN=intersystems-Instance-MyNode:MyInstance-Routine-INTEGRIT,OU=Groups,DC=intersystems,DC=com
```

LDAP Group Name Configuration

InterSystems IRIS allows you to further configure LDAP group names. The following sections describe the default configuration, the configurable properties, and the procedure to change them.

Default Group Name Configuration

By default, LDAP group names use the following syntax:

```
intersystems-Role-RoleName
intersystems-Routine-RoutineName
intersystems-Namespace-NamespaceName
intersystems-Group-GroupName-Role-RoleName
intersystems-Group-GroupName-Routine-RoutineName
intersystems-Group-GroupName-Namespace-NamespaceName
intersystems-Instance-InstanceName-Role-RoleName
intersystems-Instance-InstanceName-Routine-RoutineName
intersystems-Instance-InstanceName-Namespace-NamespaceName
```

Group Name Properties

Group names consist of the following configurable properties:

- *OrganizationID* — Default intersystems. Replace the intersystems segment of the group name with a user-defined or empty string. For example, if set to OrgABC, then the group name becomes:

```
OrgABC-Role-RoleName
OrgABC-Group-GroupName-Routine-RoutineName
OrgABC-Instance-InstanceName-Namespace-NamespaceName
```

If set to the empty string, then the group name becomes:

```
Role-RoleName
Group-GroupName-Routine-RoutineName
Instance-InstanceName-Namespace-NamespaceName
```

- *DelimiterID* — Default hyphen (-). This is the delimiter between segments in the group name. For example, if set to underscore (_), then the group name becomes:

```
intersystems_Role_RoleName
intersystems_Group_GroupName_Routine_RoutineName
intersystems_Instance_InstanceName_Namespace_NamespaceName
```

- *GroupID* — Default Group. For example, if set to SystemGrouping, then the group name becomes:

```
intersystems-SystemGrouping-GroupName-Role-RoleName
```


intersystems-SystemGrouping-GroupName-Routine-RoutineName

intersystems-SystemGrouping-GroupName-Namespace-NamespaceName

- *InstanceID* — Default Instance. For example, if set to *SystemInstance*, then the group name becomes:

intersystems-SystemInstance-InstanceName-Role-RoleName

intersystems-SystemInstance-InstanceName-Routine-RoutineName

intersystems-SystemInstance-InstanceName-Namespace-NamespaceName

- *RoleID* — Default Role. For example, if set to *SystemRole*, then the group name becomes:

intersystems-SystemRole-RoleName

intersystems-Group-GroupName-SystemRole-RoleName

intersystems-Instance-InstanceName-SystemRole-RoleName

- *NamespaceID* — Default Namespace. For example, if set to *SystemNamespace*, then the group name becomes:

intersystems-SystemNamespace-NamespaceName

intersystems-Group-GroupName-SystemNamespace-NamespaceName

intersystems-Instance-InstanceName-SystemNamespace-NamespaceName

- *RoutineID* — Default Routine. For example, if set to *SystemRoutine*, then the group name becomes:

intersystems-SystemRoutine-RoutineName

intersystems-Group-GroupName-SystemRoutine-RoutineName

intersystems-Instance-InstanceName-SystemRoutine-RoutineName

Procedure for Changing Properties

To change these properties:

1. In the Management Portal, go to the **Security LDAP Configurations** page (**Management Portal > System Administration > Security > System Security > LDAP Configuration**).
2. To edit a configuration, click on its name.
3. On this page, you can edit the *OrganizationID* property. Click on **Advanced Settings** to view and edit the rest of the properties.
4. Click **Save** at the top of the page to save your changes.

Mix Different Kinds of Groups

You can use universal groups in conjunction with single-instance or multiple-instance roles.

For example, suppose you:

- Have an application on multiple instances
- Are using universal groups
- Have a user named *UserOne* who can run the application on all instances, but cannot use it as an administrator on any machine

You would like for *UserOne* to:

- Continue to be able to run the application on all instance

- Additionally, to be able to administer the application on a particular instance, called APPTTEST, on a particular machine, called Test

To do this:

1. Set the authorization instance ID on the APPTTEST instance on the Test machine to Test:APPTTEST
2. Create the following group on the LDAP server:
`intersystems-Instance-Test_APPTTEST-Role-Administrator`
3. Assign this group to UserOne on the LDAP server
4. Create the Administrator role on the APPTTEST instance on the Test machine and grant it administrative access

You can also mix authorization groups in other ways. For example, if UserTwo has **%All** permission on all the instances authenticating to the LDAP server, you can give UserTwo exclusive administrative permission on an instance called SECRET on a machine called Server10. To do this, disable **Allow universal groups access** and then go through the process of assigning an `intersystems-Instance-Server10_SECRET-Role-Administrator` to that user.

Nested Groups

On an Active Directory LDAP server, LDAP groups include support for what are known as *nested groups*. A nested group is a group that is a member of a parent group, which means that all the users who are members of the nested group are implicitly members of the parent group. For example, suppose that there are two LDAP groups defined, known as *ABC* and *DEF*. You can make *ABC* a nested group within *DEF*; this means that, if a user is a member of *ABC*, then they are also a member of *DEF* without explicitly assigning the user to that group.

When searching for a user's nested groups, InterSystem IRIS returns only groups that are defined as Security Groups on the LDAP server. If you are using nested groups, ensure that any group used as a role for an InterSystems IRIS system is created as a Security Group.

Note: Systems which do not use nested groups will return both Security and Distribution groups.

How LDAP Groups Regulate Access to InterSystems IRIS

Through their LDAP groups, users receive roles along with a default namespace and a default routine. If the user's granted roles lack sufficient privilege for any required point of access for an instance, the user then is denied access to that instance; for example, if a user lacks sufficient privilege to use their default routine, that user is denied access.

The following rules also apply:

- If a user is assigned to a group for a role, but that role is not defined on the instance where the user is logging in, then the user does not have that role on that instance.
- If a user is assigned to a group for a default routine, but that routine is not defined on the instance where the user are logging in, then the user cannot connect to the instance.
- If a user is assigned to a group for a default namespace, but that namespace is not defined on the instance where the user are logging in, then the user cannot connect to the instance.

Configure LDAP Authorization with Operating System–Based Authentication

Topics include:

- [Operating System LDAP Authentication](#)
- [Enable OS/LDAP for an InterSystems IRIS Instance](#)
- [Enable OS/LDAP for the %Service_Console and %Service_Terminal Services](#)
- [OS/LDAP with a Single Domain and Multiple Domains](#)

- [Configure OS/LDAP with Multiple Domains for Simplified Prompting](#)

Operating System LDAP Authentication

InterSystems IRIS allows you to configure your system to support operating system–based authentication, and then to perform authorization via LDAP. This is known as *Operating System LDAP authorization* or *OS/LDAP*. It allows a user to authenticate to InterSystems IRIS using credentials from the operating system login and then to have their authorization information retrieved from an LDAP server. Operating system LDAP authorization is available in the Console on Windows and in the Terminal and on UNIX®, Linux, and macOS.

To configure OS/LDAP:

1. [Enable OS-based authentication with LDAP authorization for an InterSystems IRIS instance.](#)
2. As with standard LDAP authentication, [set up a role that is required in order to be able to log in to the instance.](#)
3. [Enable OS/LDAP for the %Service_Console and %Service_Terminal services.](#)
4. Configure authorization. This occurs in the same manner as that which accompanies LDAP authentication, as described in [Configure LDAP Authorization for InterSystems IRIS.](#)
5. If you are using [multiple domains](#), optionally [configure OS/LDAP for simplified prompting.](#)

Enable OS/LDAP for an InterSystems IRIS Instance

To use OS/LDAP, first enable it for the instance:

1. From the Management Portal home page, go to the **Authentication/Web Session Options** page (**System Administration** > **Security** > **System Security** > **Authentication/Web Session Options**).
2. On the **Authentication/Web Session Options** page, select **Allow Operating Systems LDAP authentication**.
3. Click **Save** to apply the changes.

Enable OS/LDAP for the %Service_Console and %Service_Terminal Services

To enable OS/LDAP for the instance’s relevant services or applications:

1. With LDAP authentication enabled for the instance, an **Operating System LDAP Authorization** check box appears on the **Edit Service** page for **%Service_Console** and **%Service_Terminal**, which are the services that support OS/LDAP.
2. Enable LDAP authentication for those services, as appropriate.

OS/LDAP with a Single Domain and Multiple Domains

OS/LDAP supports the use of a single domain or [multiple domains](#).

When InterSystems IRIS is configured to support only a single domain:

1. The system prompts the user for a username and password for the first login.
2. For subsequent logins, there is no prompt because the operating system has already authenticated the user.

When InterSystems IRIS is configured to support multiple domains:

1. The system prompts the user for a username and password for the first login.
2. For subsequent logins, the operating system prompts for a username and password by default. You can configure InterSystems IRIS to prevent this prompting; see the next section.

Configure OS/LDAP with Multiple Domains for Simplified Prompting

If you are using OS/LDAP and multiple domains, you can configure the instance for simplified prompting. By default, users are prompted for a username and password at every login. You can configure InterSystems IRIS so that there is only a username/password prompt when a user first logs in, and that subsequent connections are authenticated without prompting.

To configure InterSystems IRIS for this behavior:

1. For each user, create the environment variable `ISC_LDAP_CONFIGURATION` with a value of the domain in which the user is authenticating.
2. For each domain in which users are authenticating:
 - a. Ensure that there is an [LDAP configuration](#) or create one.
 - b. For that LDAP configuration, select the **Allow ISC_LDAP_CONFIGURATION environment variable** check box, which enables use of the environment variable.

Configure Authorization with LDAP Attributes

For LDAP authorization, InterSystems recommends the use of LDAP groups. However, InterSystems also supports authorization using LDAP attributes. There are three registered OIDs that are available for use with an LDAP schema to store authorization information. Each has its own dedicated purpose:

- *intersystems-Namespace* — The name of the user's default namespace (OID 1.2.840.113556.1.8000.2448.2.1).
- *intersystems-Routine* — The name of the user's default routine (OID 1.2.840.113556.1.8000.2448.2.2).
- *intersystems-Roles* — The name of the user's login roles (OID 1.2.840.113556.1.8000.2448.2.3).

To use these attributes, the procedure on the LDAP server is:

1. Enable the attributes for use. To do this, modify the value of *objectClass* field in the LDAP schema by appending the *intersystemsAccount* value to its list of values. (*intersystemsAccount* has an LDAP OID of 1.2.840.113556.1.8000.2448.1.1.)
2. Add the fields (as few or as many as required) to the schema.
3. Populate their values for the entries in the LDAP database.

Note: It is not required to use the registered LDAP schema names. In fact, you may use existing attributes from your LDAP schema.

For example, with a UNIX® LDAP server, to define the schema for using LDAP authentication with InterSystems IRIS, use the content that appears in the following definitions:

```
# Attribute Type Definitions

attributetype ( 1.2.840.113556.1.8000.2448.2.1 NAME 'intersystems-Namespace'
    DESC 'InterSystems Namespace'
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.5 SINGLE-VALUE )

attributetype ( 1.2.840.113556.1.8000.2448.2.2 NAME 'intersystems-Routine'
    DESC 'InterSystems Routine'
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{128} SINGLE-VALUE )

attributetype ( 1.2.840.113556.1.8000.2448.2.3 NAME 'intersystems-Roles'
    DESC 'InterSystems Roles'
    EQUALITY caseIgnoreMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

# Object Class Definitions

objectclass ( 1.2.840.113556.1.8000.2448.1.1
    NAME 'intersystemsAccount'
    SUP top
    AUXILIARY
```

```
DESC 'Abstraction of an account with InterSystems attributes'
MAY (
    intersystems-Routine $
    intersystems-Namespace $
    intersystems-Roles
)
)
```

This content goes to two locations:

- Place it in the `intersystems.schema` file in the `/etc/openldap/schema/` directory.
- Include it, along with any other content, in the `/etc/openldap/slapd.conf` file.

Other LDAP Topics

Create a Secure Outbound LDAP Connection

While this document primarily concerns using LDAP for authentication and authorization when connecting to InterSystems IRIS, you may also connect from InterSystems IRIS to an LDAP server. To establish a secure outbound connection to an LDAP server, InterSystems IRIS includes support for TLS. For more information on this topic, see the class documentation for `%SYS.LDAP`, in the content for the `Init` method.

Use the LDAP APIs

The `%SYS.LDAP` class supports LDAP programmatically.

If you are using the InterSystems IRIS LDAP APIs with certificates on UNIX® and need detailed debugging information, you may wish to use the `ldapsearch` program that is part of the [OpenLDAP](#) package. Once you have corrected any problems with certificates, you can use the [test configuration](#) tool to verify that the connection is functioning. The `ldapsearch` program may also be useful for debugging other LDAP connection problems.

How Various LDAP Actions Occur

This section describes what occurs during certain processes associated with LDAP authentication and authorization:

- [How LDAP Performs Authentication and Authorization](#)
- [How LDAP Looks Up the Target User in Its Database](#)
- [How an Instance Checks and Removes Local Accounts Based on LDAP Account Conditions](#)

How LDAP Performs Authentication and Authorization

When a user attempts to authenticate to an instance of InterSystems IRIS that uses LDAP authentication, the process is:

1. The user is prompted for a user name and password. This user, who is trying to authenticate, is known as the *target user*.
2. InterSystems IRIS establishes a connection to the LDAP server using the values specified for the **LDAP username to use for searches** and **LDAP username password**. This user, who has privileges to search the LDAP database so that InterSystems IRIS can retrieve information, is known as the *search user*.
3. Once the connection is established, the next step is to [look up the target user in the LDAP database](#) using the **LDAP Unique search attribute**.
4. If the target user is found in the LDAP database, it retrieves the attributes associated with the user, such as the user's roles, namespace, and routine.
5. InterSystems IRIS then attempts to authenticate the user to the LDAP database, using the user name and password provided in step 1.
6. If authentication succeeds, authorization occurs on the LDAP server (either via [group assignment](#) or [attributes](#)). The user can then interact with InterSystems IRIS based on the privileges associated with their roles and any publicly

available resources. The user's properties are displayed read-only in the Management Portal and are not editable from within InterSystems IRIS.

How LDAP Looks Up the Target User in Its Database

Once InterSystems IRIS has established a connection to the LDAP server as the search user, it next retrieves information about the target user. To do this, InterSystems IRIS checks the username provided at login against values in the LDAP database for the *LDAP Unique search attribute*. The name of this attribute is often "sAMAccountName" for an Active Directory LDAP server and "uid" for an OpenLDAP server.

Once InterSystems IRIS has located the user, it retrieves attribute information. It retrieves information about every named attribute in the InterSystems IRIS LDAP configuration fields (described in [Create or Modify an LDAP Configuration](#)), and it retrieves all values associated with each attribute. Note that InterSystems IRIS retrieves all values associated with all attributes specified for the user in the InterSystems IRIS LDAP configuration fields; it is not possible to configure it to retrieve only a subset of these.

How an Instance Checks and Removes Local Accounts Based on LDAP Account Conditions

InterSystems IRIS removes a user account on the local instance when the account meets any of the following conditions:

- The LDAP account no longer exists
- The LDAP account is disabled
- On Active Directory only, the LDAP account has the flag set to require a password change
- On Active Directory only, the LDAP account is expired

InterSystems IRIS checks for these conditions and removes accounts under the following circumstances:

- When a user attempts to log in to an InterSystems IRIS instance, the instance checks the user's LDAP account. If any of the specified conditions are true for the LDAP account, InterSystems IRIS removes the local user account.
- As a result of the [SecurityScan](#) task. InterSystems IRIS comes with this task; run it to determine if any of these conditions are true for the LDAP account associated with any local user account. If so, InterSystems IRIS removes the local user account.

OAuth 2.0 and OpenID Connect

Overview of OAuth 2.0 and OpenID Connect

This section provides a brief overview of OAuth 2.0 authorization framework and OpenID Connect. [Another page](#) introduces InterSystems IRIS® support for OAuth 2.0 and OpenID Connect.

Basics

The OAuth 2.0 authorization framework enables a third-party application (generally known as a client) to obtain limited access to an HTTP service (a resource). The access is limited; the client can obtain only specific information or can use only specific services. An authorization server either orchestrates an approval interaction or directly gives access. OpenID Connect extends this framework and adds authentication to it.

Roles

The OAuth 2.0 framework defines four roles:

- *Resource owner* — Usually a user.
- *Resource server* — A server that hosts protected data and/or services.
- *Client* — An application that requests limited access to a resource server. This can be a client-server application or can be an application that has no server (such as a JavaScript application or mobile application).
- *Authorization server* — A server that is responsible for issuing access tokens, with which the client can access the resource server. This server can be the same application as the authorization server but can also be a different application.

The client, resource server, and authorization server are known to each other, by prior arrangement. An authorization server has a registry of clients, which specifies the client servers and resource servers that can communicate with it. When it registers a client, an authorization server generates a *client ID* and a *client secret*, the latter of which must be kept secret. Depending on how the client will communicate with the authorization server, the client server might need the client secret; in that case, it is necessary to convey the client secret securely to the client server. In some scenarios (such as a JavaScript application), it is impossible for the client to protect the client secret; in these scenarios, the client must communicate with the authorization server in a way that does not require the client secret.

Access Tokens

An *access token* contains information about the identity of the user or client, as well as metadata such as an expiration date, expected issuer name, expected audience, scope, and so on.

The general purpose of an access token is to enable a client to access specific data or services available via HTTP at a resource server. In the overall flow, the client application requests an access token from the authorization server. After receiving this token, the client uses the access token within HTTP requests to the resource server. The resource server returns the requested information only when it receives requests that contain a valid access token.

An access token can also be revoked, if the authorization server supports this.

Forms of Access Tokens

InterSystems IRIS supports two forms of access tokens:

- JSON Web Tokens (JWTs). A *JWT* is a JSON object. A JWT can be digitally signed, encrypted, or both.

Note that one kind of JWT is an ID token; this is specific to OpenID Connect.

A JWT can be signed, encrypted, or both.

- *Opaque access tokens* (also known as *reference tokens*). This form of access token is just the identifier of a token that is stored elsewhere, specifically on the authorization server. The identifier is a long, random string, intended to be very difficult to guess.

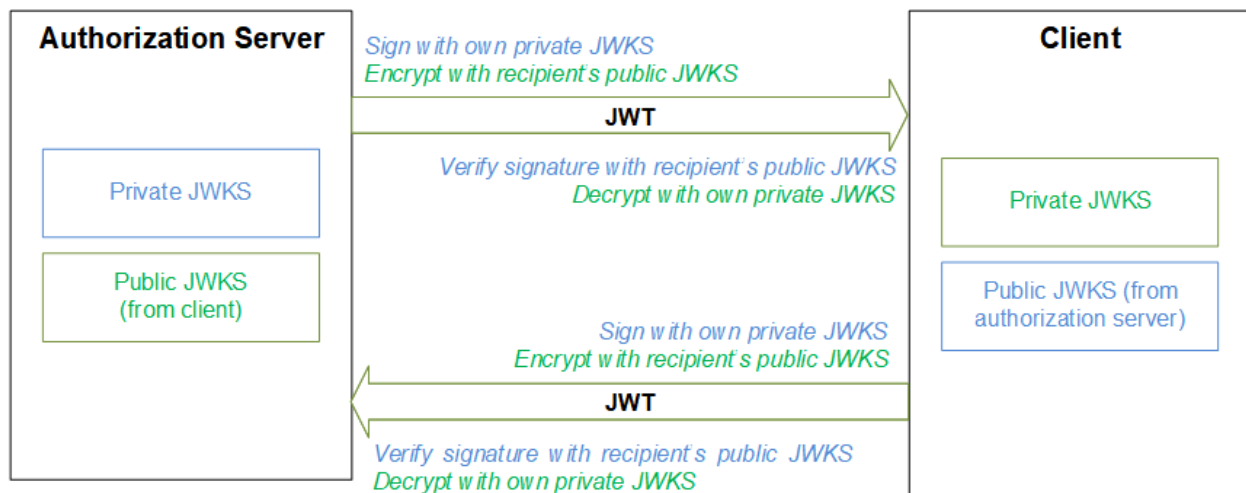
Claims

An access token contains a set of *claims* that communicate the identity of the user or client, or that communicate metadata such the token's expiration date, expected issuer name, expected audience, scope, and so on. The OpenID Connect Core specification defines a standard set of claims, and other claims may be used as well.

JWTs and JWKSs

As noted above, a JWT can be signed, encrypted, or both. In most cases, the participants in the OAuth 2.0 framework use pairs of JWKSs (JSON web key sets) for this purpose. In any pair of JWKSs, one JWKS is private and contains all the needed private keys (per algorithm) as well as the client secret for use as a symmetric key; this JWKS is never shared. The other JWKS contains the corresponding public keys and is publicly available.

Each participant has a private JWKS and provides other participants with the corresponding public JWKS. The owner of a private JWKS uses that JWKS for signing outbound JWTs and decrypting inbound JWTs. The other parties use the corresponding public JWKS to encrypt outbound JWTs and verify signatures of inbound JWTs, as shown in the following figure:



Grant Types and Flows

In the OAuth 2.0 framework, a *grant type* specifies how the authorization server should process the request for authorization. The client specifies the grant type within the initial request to the authorization server. The OAuth 2.0 specification describes four grant types, as well as an extensibility mechanism for defining additional types. In general, each grant type corresponds to a different overall flow.

The four grant types are as follows:

- *Authorization code* — This grant type can be used only with a client application that has a corresponding server. In this grant type, the authorization server displays a login page with which the user provides a username and password; these are never shared with the client. If the username and password correspond to a valid user (and if other elements of the request are in order), the authorization server first issues an authorization code, which it returns to the client. The client then uses the authorization code to obtain an access token.

The request for the authorization code is visible in the browser, as is the response. The request for the access token, however, is a server-to-server interaction, as is that response. Thus the access token is never visible in the browser.

Proof Key for Code Exchange (PKCE) is an extension to the authorization code flow that prevents a malicious actor from obtaining an access token with an intercepted authorization code. With PKCE, the client's request for an autho-

authorization code includes an additional secret value. The authorization server saves this secret when it issues the authorization code. The client's subsequent request to exchange the authorization code for an access token must include the original secret; someone who had intercepted the authorization code would not know this secret, thereby preventing them from obtaining an access token.

- *Implicit* — As with the previously listed grant type, the authorization server displays a login page, and the client never has access to the user's credentials. However, in the implicit grant type, the client directly requests and receives an access token. This grant type is useful for pure client applications such as JavaScript clients or mobile applications.
- *Resource owner password credentials* — In this grant type, the client prompts the user for a username and password and then uses those credentials to obtain an access token from the authorization server. This grant type is suitable only with trusted applications.
- *Client credentials grant type* — In this grant type, there is no user context, and the client application is unattended. The client uses its client ID and client secret to obtain an access token from the authorization server.

[RFC 7523](#) describes an additional grant type, JWT authorization. This grant type uses of a JSON Web Token (JWT) Bearer Token to request an OAuth 2.0 access token and to authenticate the client; InterSystems IRIS supports this grant type in addition to the four in the OAuth 2.0 specification.

Note that in the OAuth 2.0 framework, in general, all HTTP requests are protected by SSL/TLS.

In addition, when a client sends a request to the authorization server, that request must be authenticated. The OAuth 2.0 specification describes the ways in which a client can authenticate the request.

Scopes

The authorization server allows the client to specify the scope of the access request using the `scope` request parameter. In turn, the authorization server uses the `scope` response parameter to inform the client of the scope of the access token issued.

OpenID Connect is an extension to the OAuth 2.0 authorization process. To request authentication, the client includes the `openid` scope value in the request to the authorization server. The authorization server returns information about the authentication in a JWT called an *ID token*. An ID token contains a specific set of claims, listed in the OpenID Connect Core specification.

Endpoints in an Authorization Server

An authorization server provides some or all of the following URLs or endpoints, which can process requests of varying kinds:

Endpoint	Purpose
Authorization endpoint	Returns an authorization code (applies only to authorization code grant type)
Token endpoint	Returns an access token
Userinfo endpoint	Returns a JSON object that contains claims about the authenticated user (applies only to OpenID Connect)
Token introspection endpoint	Returns a JSON object that contains claims determined by examining an access token
Token revocation endpoint	Revokes a token

Using an InterSystems IRIS Web Application as an OAuth 2.0 Client

This page describes how to use an InterSystems IRIS® web application as a client application that uses the [OAuth 2.0 framework](#). The focus is the scenario in which an InterSystems IRIS web application is the client of a web server/client application and uses the authorization code grant type. Also see [OAuth 2.0 Client Variations](#).

Note: When your OAuth 2.0 client is communicating with an Active Directory Federation Service (ADFS) authorization server, your client code must append a special key-value pair to the authorization endpoint. For more details, see the description of `GetAuthorizationCodeEndpoint()` in [Method Details](#).

Prerequisites for the InterSystems IRIS Client

Before starting the tasks described in this page, make sure the following items are available:

- An OAuth 2 authorization server. Later you will need to know specific details about this server. Some of the details apply when you configure the client within InterSystems IRIS:
 - Location of the authorization server (issuer endpoint)
 - Location of the authorization endpoint
 - Location of the token endpoint
 - Location of the Userinfo endpoint (if supported; see [OpenID Connect Core](#))
 - Location of the token introspection endpoint (if supported; see [RFC 7662](#))
 - Location of the token revocation endpoint (if supported; see [RFC 7009](#))
 - Whether the authorization server supports dynamic registration

Other details apply when you write the client code:

- Grant types supported by this server
 - Scopes supported by this server. For example, the server may or may not support `openid` and `profile`, which are special scopes defined by [OpenID Connect Core](#).
 - Other requirements for requests made to this server
- If the authorization server does not support dynamic client registration, the InterSystems IRIS application must be registered as a client of the OAuth 2.0 authorization server, and you must have the client ID and client secret for this client. The details depend upon the implementation of the authorization server. (If the server does support dynamic registration, you can register the client while configuring it as described in this page.)

Configuration Requirements

To use an InterSystems IRIS web application as an OAuth 2.0 client, perform the following configuration tasks:

- For the web server that is serving InterSystems IRIS, configure that web server to use SSL. It is beyond the scope of this documentation to describe how to configure a web server to use SSL.
- Create an InterSystems IRIS SSL configuration for use by the client.

This should be a client SSL configuration; no certificate is needed. The configuration is used to connect to a web server. Via this connection, the client communicates with the authorization server to obtain access tokens, call the Userinfo endpoint, call the introspection endpoint, and so on.

For details on creating SSL configurations, see [InterSystems TLS Guide](#).

Each SSL configuration has a unique name. For reference, the documentation refers to this one as `sslconfig`, but you can use any unique name.

- Create the OAuth 2.0 configuration items for the client. To do so, first create the [server description](#) and then create the [client configuration](#), as described in the subsections.

For both items, to find the needed options in the Management Portal, select **System Administration > Security > OAuth 2.0 > Client Configuration**. This page provides the options needed when you create an OAuth 2.0 configuration on a client machine (that is, on any machine other than one being used as an authorization server).

On a client machine, do not use the menu **System Administration > Security > OAuth 2.0 > Server Configuration**.

Creating a Server Description (Using Discovery)

1. In the Management Portal, select **System Administration > Security > OAuth 2.0 > Client Configuration**.

This displays a page that lists any server descriptions that are available on this instance. In any given row, the **Issuer endpoint** column indicates the issuer endpoint for the server description. The **Client Count** column indicates the number of client configurations associated with the given server description. In the last column, the **Client Configurations** link enables you to create, view, edit, and delete the associated client configurations.

2. Select **Create Server Configuration**.

The Management Portal then displays a new page where you can enter details for the server description.

3. Specify the following details:

- **Issuer endpoint** (required) — Enter the endpoint URL to be used to identify the authorization server.
- **SSL/TLS configuration** (required) — Select the SSL/TLS configuration to use when making the dynamic client registration request.
- **Registration access token** — Optionally enter the initial registration access token to use as a bearer token to authorize the dynamic client registration request.

4. Select **Discover and Save**.

InterSystems IRIS then communicates with the given authorization server, retrieves information needed in the server description, and then saves that information.

The Management Portal then redisplay the list of server descriptions.

Manually Creating a Server Description (No Discovery)

To manually create a server description (rather than using discovery), first display the server description page (steps 1 and 2 above) and then select **Manual**. Then the page displays a larger set of options, as follows:

- **Issuer endpoint** (required) — Enter the endpoint URL to be used to identify the authorization server.
- **Authorization endpoint** (required) — Enter the endpoint URL to be used when requesting an authorization code from the authorization server.
- **Token endpoint** (required) — Enter the endpoint URL to be used when requesting an access token from the authorization server.
- **Userinfo endpoint** — Enter the endpoint URL to be used when making a Userinfo request using an access token from the authorization server for authorization.
- **Token introspection endpoint** — Enter the endpoint URL to be used when making a token introspection request using the `client_id` and `client_secret` for authorization. See [RFC 7662](#).
- **Token revocation endpoint** — Enter the endpoint URL to be used when making a token revocation request using the `client_id` and `client_secret` for authorization. See [RFC 7009](#).
- **JSON Web Token (JWT) Settings** — Specifies the source of the public keys that the client should use for signature verification and decryption of JWTs from the authorization server.

By default, the authorization server generates a pair of **JWKSs** (JSON web key sets). One JWKS is private and contains all the needed private keys (per algorithm) as well as the client secret for use as a symmetric key; this JWKS is never shared. The other JWKS contains the corresponding public keys and is publicly available. The process of creating the server description also copies the public JWKS from the authorization server to the client for its use in signature verification and encryption of JWTs.

- **JWKS from URL** — Specify a URL that points to a public JWKS and then load the JWKS into InterSystems IRIS.
- **JWKS from file** — Select a file that contains a public JWKS and then load that file into InterSystems IRIS.
- **X509 certificate** — For details, see [Using Certificates for an OAuth 2.0 Authorization Server](#), in [Certificates and JWTs \(JSON Web Tokens\)](#).

To access any of these options, first select **Source other than dynamic registration**.

Specify these values and then select **Save**.

Configuring and Dynamically Registering a Client

This section describes how to create a client configuration and dynamically register the client.

1. In the Management Portal, select **System Administration > Security > OAuth 2.0 > Client Configuration**.

The Management Portal displays the list of server descriptions.

2. Click the **Client Configurations** link in the row for the [server description](#) with which this client configuration should be associated.

The Management Portal then displays the list of client configurations associated with the server description. This list is initially empty.

3. Click **Create Client Configuration**.

The Management Portal then displays a new page where you can enter details.

4. On the **General** tab, specify the following details:

- **Application name** — Specify a short name for the application.
- **Client name** — Specify the client name to display to the end user.
- **Description** — Specify an optional description of the application.
- **Enabled** — Optionally clear this check box if you want to prevent this application from being used.
- **Client Type** — Select one of the following:

- **Confidential** — Specifies that the client is a confidential client, per [RFC 6749](#).

This page primarily discusses the scenario in which the client uses the authorization code grant type. For this scenario, specify **Client Type** as **Confidential**. For other grant types, see [Variations](#).

- **Public** — Specifies that the client is a public client, per [RFC 6749](#).
- **Resource server** — Specifies that the client is a resource server which is not also a client.

- **SSL/TLS configuration** — Select the SSL configuration you created for use by the client (for example, `sslconfig`).
- **The client URL to be specified to the authorization server to receive responses** — Specify the URL of the internal destination required for an InterSystems IRIS OAuth 2.0 client. At this destination, the access token is saved and then the browser is further redirected back to the client application.

To specify this URL, enter values for the following options:

- **Host name** — Specify the host name or IP address of the authorization server.
- **Port** — Specify this if needed to accommodate any changes in the [Web Gateway](#) configuration.
- **Prefix** — Specify this if needed to accommodate any changes in the [Web Gateway](#) configuration.

The resulting URL has the following form:

```
https://hostname:port/prefix/csp/sys/oauth2/OAuth2.Response.cls
```

If you omit **Port**, the colon is omitted. Similarly, if you omit **Prefix**, there is only one slash between `hostname:port` and `csp`. (Also if the **Use TLS/SSL** option is cleared, the URL starts with `http` rather than `https`.)

- **Use TLS/SSL** — Select this option, unless there is a good reason not to use TLS/SSL when opening the redirect page.
- **Front Channel Logout URL** — Optionally specify the HTTP-based front channel logout URL. The server registers this URL and uses it log users out on the client. To create a client that does not support front channel logout, leave the URL empty. The box above this field displays the specified URL and appends 'IRISLogout=end' to it.

Note: For an InterSystems IRIS client to support front channel logout, the **Session Cookie Scope** of the client application to `None`. For details on configuring application settings, see [Create and Edit Applications](#).

- **Required grant types** — Specify the OAuth 2.0 grant types that the client will restrict itself to using.
- **Authentication type** — Select the type of authentication (as specified in [RFC 6749](#) or [OpenID Connect Core section 9](#)) to be used for HTTP requests to the authorization server. Select one of the following: **none**, **basic**, **form encoded body**, **client secret JWT**, or **private key JWT**.
- **Require iss and sid query parameters when auth server calls logout URL** — Select this option to require the `iss` (issuer) and `sid` (session ID) query parameters when the authorization server calls the front channel logout URL.
- **Authentication signing algorithm** — Select the algorithm that must be used for signing the JWTs used to authenticate this client at the token endpoint (if the authentication type is **client secret JWT** or **private key JWT**). If you do not select an option, any algorithm supported by the OpenID provider and the relying party may be used.

5. On the **Client Information** tab, specify the following details:

- **Logo URL** — URL of the logo for the client application.
- **Client home page URL** — URL of the home page for the client application.
- **Policy URL** — URL of the policy document for the client application.
- **Terms of service URL** — URL of the terms of service document for the client application.
- **Default scope** — Specify the default scope, as a blank separated list, for access token requests. This default should be consistent with the scopes permitted by the authorization server.
- **Contact emails** — Comma-separated list of email addresses suitable for use in contacting those responsible for the client application.
- **Default max age** — Specify the default maximum authentication age, in seconds. If you specify this option, the end user must be actively re-authenticated when the maximum authentication age is reached. The `max_age` request parameter overrides this default value. If you omit this option, there is no default maximum authentication age.

6. On the **JWT Settings** tab, specify the following details:

- **Create JWT Settings from X509 credentials** — Select this option if, for signing and encryption, you want to use the private key associated with a certificate; in this case, also see [Using Certificates for an OAuth 2.0 Client](#), in [Certificates and JWTs \(JSON Web Tokens\)](#).

Note: InterSystems expects that the option **Create JWT Settings from X509 credentials** will rarely be used, and that instead customers use the default behavior described next.

If you leave this option clear, the system generates a pair of **JWKSs** (JSON web key sets). One JWKS is private and contains all the needed private keys (per algorithm) as well as the client secret for use as a symmetric key;

this JWKS is never shared. The other JWKS contains the corresponding public keys and is publicly available. The dynamic registration process also copies the public JWKS to the authorization server, so that the authorization server can encrypt and verify signatures of JWTs from this client.

- **Signing algorithm** — Select the signing algorithm used to create signed JWTs. Or leave this blank if JWTs are not to be signed.
 - **Encryption algorithm** — Select the encryption algorithm used to create encrypted JWTs. Or leave this blank if JWTs are not to be encrypted. If you select a value, you must also specify **Key algorithm**.
 - **Key algorithm** — Select the key management algorithm used to create encrypted JWTs. Or leave this blank if JWTs are not to be encrypted.
7. If the authorization server supports dynamic registration, double-check all the data you have entered and then press **Dynamic Registration and Save**. InterSystems IRIS then contacts the authorization server, registers the client, and obtains the client ID and client secret.

If the authorization server does not support dynamic registration, see the following subsection.

Configuring a Client (No Dynamic Registration)

If the authorization server does not support dynamic registration, then do the following instead of the last step above:

1. Select the **Client Credentials** tab and specify the following details:
 - **Client ID** — Enter the client ID as provided by the authorization server.
 - **Client secret** — Enter the client secret as provided by the authorization server. This value is required if the **Client Type** is **Confidential**.

This page primarily discusses the scenario in which the client uses the authorization code grant type. For this scenario, specify a value for **Client secret**. For other grant types, see [Variations](#).

Do not enter values for **Client ID Issued At**, **Client Secret Expires At**, and **Registration Client Uri**.

2. Select **Save**.

Outline of Code Requirements

Note: This section describes the code needed when the client uses the authorization code grant type when requesting tokens. For other grant types, see [Variations](#).

In order for an InterSystems IRIS web application to act as OAuth 2.0 client, this web application must use logic like the following:

1. Obtain an access token (and if needed, an ID token). See [Obtaining Tokens](#).
2. Examine the access token and (optionally, an ID token) to determine whether the user has the necessary permissions to use the requested resource. See [Examining the Tokens](#).
3. If appropriate, call the resource server as described in [Adding an Access Token to an HTTP Request](#).

The following sections provide information on these steps.

Obtaining Tokens

Note: This section provides information on the code needed when the client uses the authorization code grant type when requesting tokens. By default, this authorization code grant type includes the Proof Key for Code Exchange (PKCE) extension. For other grant types and authorization code without PKCE, see [Variations](#).

To obtain tokens, use steps like the following to obtain tokens. The [subsection](#) provides details on the methods discussed here.

1. Call the **IsAuthorized()** method of the %SYS.OAuth2.AccessToken class. For this, you will need to first determine the desired scope or scopes for the access token.

For example:

ObjectScript

```
set myscope="openid profile scopel scope2"
set isAuth##class(%SYS.OAuth2.AccessToken).IsAuthorized("myclient",,myscope,
    .accessToken,.idToken,.responseProperties,.error)
```

This method checks to see whether an access token has already been saved locally.

2. Check to see if the *error* argument has returned an error and then handle that error appropriately. Note that if this argument contains an error, the function *\$ISOBJECT()* will return 1; otherwise *\$ISOBJECT()* will return 0.

ObjectScript

```
if $isobject(error) {
    //error handling here
}
```

3. If **IsAuthorized()** returns 1, skip to [Examining the Tokens](#).
4. Otherwise, call the **GetAuthorizationCodeEndpoint()** method of the %SYS.OAuth2.Authorization class. For this, you will need the following information:
 - Complete URL that the authorization server should redirect to, after it returns an access token. This is the *client's redirect page* (which can be the same as the original page, or can be different).
 - The scope or scopes of the request.
 - Any parameters to be included with the request. For example, you may need to pass the `claims` parameter.

For example:

ObjectScript

```
set scope="openid profile scopel scope2"
set redirect="https://localhost/csp/openid/SampleClientResult.csp"

set url=##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint("myclient",
    scope,redirect,.properties,.isAuthorized,.sc)
if $$$ISERR(sc) {
    //error handling here
}
```

This method returns the full URL, including query parameters, of the internal destination required for an InterSystems IRIS OAuth 2.0 client.

To modify the default Proof Key for Code Exchange (PKCE) behavior for this method, see details about the *properties* argument in [Method Details](#).

5. Provide an option (such as a button) that opens the URL returned by **GetAuthorizationCodeEndpoint()**, thus enabling the user to authorize the request.

At this internal URL, which is never visible to users, InterSystems IRIS obtains an authorization code, exchanges that for an access token, and then redirects the browser to the client's redirect page.

Method Details

This subsection provides the details on the methods described in the previous subsection.

IsAuthorized()

Location: This method is in the class %SYS.OAuth2.AccessToken.

```
ClassMethod IsAuthorized(applicationName As %String,
                        sessionId As %String,
                        scope As %String = "",
                        Output accessToken As %String,
                        Output IDToken As %String,
                        Output responseProperties,
                        Output error As %OAuth2.Error) As %Boolean
```

This method returns 1 if there is a locally stored access token for this client and this session, *and* if that access token authorizes all the scopes given by the *scope* argument. (Note that this method looks for the access token in the IRISYS database, and that tokens are removed automatically after they have expired.)

Otherwise the method returns 0.

The arguments are as follows:

- *applicationName* is the name of the client application.
- *sessionId* specifies the session ID. Specify this only if you want to override the default session (%session.SessionId).
- *scope* is a space-delimited list of scopes, for example: "openid profile scope1 scope2"
Note that openid and profile are special scopes defined by [OpenID Connect Core](#).
- *accessToken*, which is returned as output, is the access token, if any.
- *IDToken*, which is returned as output, is the ID token, if any. (This applies only if you are using [OpenId Connect](#), specifically if the request used the scope openid.) Note that an ID token is a JWT.
- *responseProperties*, which is returned as output, is a multidimensional array that contains any parameters of the response. This array has the following structure:

Array node	Array value
<i>responseProperties(parametername)</i> where <i>parametername</i> is the name of a parameter (such as token_type or expires_in)	Value of the given parameter.

- *error*, which is returned as output, is either (when there is no error) an empty string or (in the case of error) an instance of %OAuth2.Error containing error information.

%OAuth2.Error has three string properties: Error, ErrorDescription, and ErrorUri.

GetAuthorizationCodeEndpoint()

Location: This method is in the class %SYS.OAuth2.Authorization.

```
ClassMethod GetAuthorizationCodeEndpoint(applicationName As %String,
                                        scope As %String,
                                        redirectURL As %String,
                                        ByRef properties As %String,
                                        Output isAuthorized As %Boolean,
                                        Output sc As %Status,
                                        responseMode As %String,
                                        sessionId As %String = "") As %String
```

This method returns the URL, with all needed query parameters, of the local, internal page that InterSystems IRIS uses to request the authorization code. (Note that this page is never visible to users.)

The arguments are as follows:

- *applicationName* is the name of the client application.

- *scope* is a space-delimited list of scopes for which access is requested, for example: "scope1 scope2 scope3"

The default is determined by the [client configuration](#) for the given *applicationName*.

- *redirectURL* is the full URL of the client's redirect page, the page to which the authorization server should redirect the browser after returning the access token to the client.
- *properties*, which is passed by reference, is a multidimensional array that contains any parameters to be added to the request. This array must have the following structure:

Array node	Array value
<i>properties(parametername)</i> where <i>parametername</i> is the name of a parameter	<p>Value of the given parameter. The value can be a scalar value, an instance of a dynamic object, or the UTF-8 encoded serialized form of a dynamic object.</p> <p>Use a dynamic object if you want the request to include a parameter whose value is a JSON object; a scenario is the <code>claims</code> parameter that is defined by OpenID Connect. For details on dynamic objects, see Using JSON.</p> <p>To use the <code>request</code> or <code>request_uri</code> parameter, see the section Passing Request Objects as JWTs.</p> <p>You can use the <code>code_verifier</code> parameter to modify the secret PKCE value sent to the authorization server. By default, the PKCE secret is generated from a random 43 character string using a SHA-256 hash. To generate the secret from a custom string (for example, if you want to use 128 characters), set the custom value to the <code>code_verifier</code> parameter.</p>

- *isAuthorized*, which is returned as output, equals 1 if there is a locally stored access token for this client and this session (scope is not checked). This parameter equals 0 otherwise. There is no need to check this output argument, because we have just called the **IsAuthorized()** method.
- *sc*, which is returned as output, contains the status code set by this method.
- *responseMode* specifies the mode of the response from the authorization server. This can be "query" (the default), "fragment" or "form_post". The default is almost always appropriate.
- *sessionId* specifies the session ID. Specify this only if you want to override the default session (`%session.SessionId`).

Note: An Active Directory Federation Services (AFDS) server expects the authorization endpoint URL to include the key-value pair `resource=urn:microsoft:userinfo`. You can use the *properties* argument of `GetAuthorizationCodeEndpoint` to append this key-value pair to the end of the URL that is defined in the server description. You should avoid using the Management Portal to modify the authorization endpoint to include this information. Rather, use the following code to modify the *properties* argument before calling the `GetAuthorizationCodeEndpoint` method:

```
set properties("resource") = "urn:microsoft:userinfo"
set url = ##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint(appName, scopes,
clientRedirectURI, .properties, .isAuthorized, .sc)
```

Also see [Variation: Performing the Redirect within OnPreHTTP](#).

Examining the Token(s)

After the client receives an access token (and, optionally, an ID token), the client should perform additional checks to determine whether the user has the necessary permissions to use the requested resource. To perform this examination, the client can use the methods described here to obtain additional information.

ValidateIDToken()

Location: This method is in the class %SYS.OAuth2.Validation.

```
ClassMethod ValidateIDToken(applicationName As %String,  
                           IDToken As %String,  
                           accessToken As %String,  
                           scope As %String,  
                           aud As %String,  
                           Output jsonObject As %RegisteredObject,  
                           Output securityParameters As %String,  
                           Output sc As %Status) As %Boolean
```

This method validates the signed OpenID Connect ID token (*IDToken*) and creates an object (*jsonObject*) to contain the properties of the ID token. To validate the ID token, the method checks the audience (if *aud* is specified), endpoint (must match that specified in server description), and scope (if *scope* is specified), and signature. The method also makes sure the ID token has not expired.

This method also validates the access token (*accessToken*) based on the *at_hash* property of the ID token.

This method returns 1 if the ID token is valid or returns 0 otherwise. It also returns several arguments as output.

The arguments are as follows:

- *applicationName* is the name of the client application.
- *IDToken* is the ID token.
- *accessToken* is the access token.
- *scope* is a space-delimited list of scopes, for example: "scope1 scope2 scope3"
- *aud* specifies the audience that is using the token. If the token has an associated *aud* property (usually because the audience was specified when requesting the token), then *aud* is matched to the token audience. If *aud* is not specified, then no audience checking takes place.
- *jsonObject*, which is returned as output, is a dynamic object that contains the properties of the *IDToken*. Note that an ID token is a JWT. For details on dynamic objects, see [Using JSON](#).
- *securityParameters*, which is returned as output, is a multidimensional array that contains security information taken from the header, for optional additional use in verifying signatures, decrypting, or both. See the *securityParameters* argument for **ValidateJWT()**.
- *sc*, which is returned as output, contains the status code set by this method.

If this method returns success (1), examine *jsonObject*, and use the contained claims as needed to determine whether to allow access to the requested resource. Use *securityParameters* if needed.

GetUserinfo()

Location: This method is in the class %SYS.OAuth2.AccessToken.

```
ClassMethod GetUserinfo(applicationName As %String,  
                        accessToken As %String,  
                        IDTokenObject As %RegisteredObject,  
                        Output jsonObject As %RegisteredObject,  
                        Output securityParameters As %String) As %Status
```

This method sends the access token to the Userinfo endpoint, receives a response that contains claims, and creates an object (*jsonObject*) that contains the claims returned by that endpoint. If the response returns a JWT, then the response is decrypted and the signature is checked before *jsonObject* is created. If the argument *IDTokenObject* is specified, the method also verifies that the sub claim from the User info endpoint matches the sub claim in *IDTokenObject*.

The request is authorized using the specified access token.

The arguments are as follows:

- *applicationName* is the name of the client application.
- *accessToken* is the access token.
- *IDTokenObject* (optional), is a dynamic object containing an ID token. For details on dynamic objects, see [Using JSON](#).
- *jsonObject*, which is returned as output, is a dynamic object that contains the claims returned by Userinfo endpoint.
- *securityParameters*, which is returned as output, is a multidimensional array that contains security information taken from the header, for optional additional use in verifying signatures, decrypting, or both. See the *securityParameters* argument for **ValidateJWT()**.

If this method returns success (1), examine *jsonObject*, and use the contained claims as needed to determine whether to allow access to the requested resource. Use *securityParameters* if needed.

Adding an Access Token to an HTTP Request

After the client application has received and examined an access token, the application can make HTTP requests to the resource server. Depending on the application, those HTTP requests may need the access token.

To add an access token to an HTTP request (as a bearer token HTTP authorization header), do the following:

1. Create an instance of %Net.HttpRequest and set properties as needed.

For details on this class, see [Sending HTTP Requests](#) in *Using Internet Utilities*.

2. Call the AddAccessToken() method of %SYS.OAuth2.AccessToken, which adds the access token to the HTTP request. This method is as follows:

```
ClassMethod AddAccessToken(httpRequest As %Net.HttpRequest,
                           type As %String = "header",
                           sslConfiguration As %String,
                           applicationName As %String,
                           sessionId As %String) As %Status
```

This method adds the bearer access token associated with the given application and session to the resource server request as defined by [RFC 6750](#). The arguments are as follows:

- *httpRequest* is the instance of %Net.HttpRequest that you want to modify.
- *type* specifies how to include the access token in the HTTP request:
 - "header" — Use the bearer token HTTP header.
 - "body" — Use form-encoded body. In this case, the request must be a POST with a form-encoded body.
 - "query" — Use a query parameter.
- *sslConfiguration* is the InterSystems IRIS SSL configuration to use for this HTTP request. If you omit this, InterSystems IRIS uses the SSL configuration associated with the [client configuration](#).
- *applicationName* is the name of the client application.

- `sessionId` specifies the session ID. Specify this only if you want to override the default session (`%session.SessionId`).

This method returns a status code, which your code should check.

3. Send the HTTP request (as described in [Sending HTTP Requests](#) in *Using Internet Utilities*. To do so, you call a method such as **Get()** or **Put()**.
4. Check the status returned by the previous step.
5. Optionally examine the HTTP response, which is available as the `HttpResponse` property of the HTTP request.
See [Sending HTTP Requests](#) in *Using Internet Utilities*.

For example:

ObjectScript

```
set httpRequest=##class(%Net.HttpRequest).%New()  
// AddAccessToken adds the current access token to the request.  
set sc=##class(%SYS.OAuth2.AccessToken).AddAccessToken(httpRequest,,"sslunittest",applicationName)  
if $$$ISOK(sc) {  
    set sc=httpRequest.Get("https://myresourceserver/csp/openid/openid.SampleResource.cls")  
}
```

Optionally Defining Delegated Authentication for the Web Client

You can optionally define delegated authentication for an InterSystems IRIS web client that is used as an OAuth 2.0 client. InterSystems IRIS provides two ways that you can do this:

- By creating and using a **ZAUTHENTICATE** routine, starting from a sample that is provided for use with OAuth 2.0. Your client code must also call `%session.Login()`.
- By creating and using a custom login page. It is also necessary to create and use a **ZAUTHENTICATE** routine (starting from the same sample that is provided for use with OAuth 2.0), but your client code does not need to call `%session.Login()`.

The following subsections give the details. A final [subsection](#) discusses the **ZAUTHENTICATE** sample.

Also see [REST Applications and OAuth 2.0](#) in *Securing REST Services* in *Creating REST Services*.

Important: If using authentication with HealthShare®, you must use the **ZAUTHENTICATE** routine provided by InterSystems and cannot write your own.

Creating and Using a ZAUTHENTICATE Routine for an OAuth 2.0 Client

To create and use a **ZAUTHENTICATE** routine for an InterSystems IRIS web client that is used as an OAuth 2.0 client, do all of the following:

- In your client code, after you call the **IsAuthorized()** method of the `%SYS.OAuth2.AccessToken` class and successfully obtain an access token, call the **Login()** method of the `%session` variable. For the username, specify the OAuth 2.0 application name; for the password, specify the web session ID.
- Create the **ZAUTHENTICATE** routine. This routine must perform basic setup for a user account, such as specifying roles and other user properties.

InterSystems provides a sample routine, `OAUTH2.ZAUTHENTICATE.mac`, that you can copy and modify. This routine is part of the Samples-Security sample on GitHub (<https://github.com/interSystems/Samples-Security>). You can download the entire sample as described in [Downloading Samples for Use with InterSystems IRIS](#), but it may be more convenient to simply open the routine on GitHub and copy its contents.

The **ZAUTHENTICATE** routine, if defined, must be in the %SYS namespace (and must be named ZAUTHENTICATE). See [Notes about the OAUTH2.ZAUTHENTICATE.mac Sample](#). For more general information on delegated authentication, see [Creating Delegated \(User-Defined\) Authentication Code](#) and [Delegated Authentication](#).

- [Enable delegated authentication](#) for the InterSystems IRIS instance on the [Authentication Options](#) page.
For information on this step and the next step, see [Delegated Authentication](#).
- Enable delegated authentication for the relevant [web application](#).

Creating and Using a Custom Login Page for an OAuth 2.0 Client

To create and use a custom login page for an InterSystems IRIS web client that is used as an OAuth 2.0 client, do all of the following:

- Create a subclass of %OAuth2.Login. In your subclass:
 - Specify the application name, scope list, and (optionally) response mode. You can specify these items in either or both of the following ways:
 - By specifying parameters of your subclass of %OAuth2.Login.
 - By overriding the **DefineParameters()** class method. In contrast to specifying parameters, this technique enables you to set these values at runtime.

The parameters are as follows:

- *APPLICATION* — This must be the application name for the application being logged in to.
- *SCOPE* — This specifies the scope list to be used for the access token request. This must be a blank-separated list of strings.
- *RESPONSEMODE* — This specifies the mode of the response. The allowed values are "query" (the default), "fragment" or "form_post".

The **DefineParameters()** class method has the following signature:

```
ClassMethod DefineParameters(Output application As %String, Output scope As %String, Output responseMode As %String)
```

This method returns the application name, scope list, and response mode as output arguments. The default implementation of this method returns the values of the *APPLICATION*, *SCOPE*, and *RESPONSEMODE* class parameters.

- In your subclass of %OAuth2.Login, also specify the properties list for the **GetAccessTokenAuthorizationCode()** call. To do so, override the **DefineProperties()** class method. This method has the following signature:

```
ClassMethod DefineProperties(Output properties As %String)
```

This method returns (as output) the *properties* array, which is a local array specifying additional properties to be included in a token request. The *properties* array has the following form:

Node	Value
<i>properties</i> (name), where <i>name</i> is the name of a parameter.	Value of the given parameter.

To add a request parameter that is a JSON object, create a properties element which is an instance of %DynamicObject. Or create a string that is the UTF-8 encoded serialized object.

To add the `request` or `request_uri` request parameters, use the `%SYS.OAuth2.Request` class to create the JWT. Then, as appropriate, set `properties("request")` equal to the JWT or set `properties("request_uri")` equal to the URL of the JWT.

- Configure the relevant [web application](#) to use the custom login page.
- Create and use a **ZAUTHENTICATE** routine as described in the [previous section](#), except for the first bullet item. (In this scenario, there is no need for the client code to call `%session.Login()`.)

Notes about the OAUTH2.ZAUTHENTICATE.mac Sample

The OAUTH2.ZAUTHENTICATE.mac sample (from <https://github.com/intersystems/Samples-Security>) supports both scenarios described in the previous subsections. In this sample, the **GetCredentials()** subroutine looks like this:

```
GetCredentials(ServiceName, Namespace, Username, Password, Credentials) Public {
  If ServiceName="%Service_WebGateway" {
    // Supply user name and password for authentication via a subclass of %OAuth2.Login
    Set Username="OAuth2"
    Set Password=$c(1,2,3)
  }
  Quit $$$OK
}
```

This subroutine is called if no username and password are provided (which is the case when the custom login page is being used). For the service `%Service_WebGateway`, this sample sets the username and password to specific values that are also used in the **ZAUTHENTICATE()** subroutine (which is called in later processing).

The **ZAUTHENTICATE()** subroutine includes the following:

```
If Username="OAuth2", Password=$c(1,2,3) {
  // Authentication is via a subclass of %OAuth2.Login that sets the query parameter CSPOAUTH2
  // with a hash value that allows GetCurrentApplication to determine the application --
  // username/password is supplied by GetCredentials.
  Set sc=##class(OAuth2.Response).GetCurrentApplication(.applicationName)
  Set sessionId=%session.SessionId
} Else {
  // If authentication is based on %session.Login, then application and session id are passed in.
  Set applicationName=Username
  Set sessionId=Password
}
```

A later step calls the **isAuthorized()** method like this:

```
Set
isAuthorized=##class(%SYS.OAuth2.AccessToken).IsAuthorized(applicationName, sessionId, .accessToken, .error)
```

If **isAuthorized()** returns 1, then later code calls the introspection endpoint and uses the information obtained there to define a user:

```
Set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection(applicationName, accessToken, .jsonObject)
...
Set Username="OAuth2"_jsonObject.sub
Set Properties("FullName")="OAuth account "_Username
Set Properties("Username")=Username
Set Properties("Password")="" // we don't really care about oauth2 account password
// Set the roles and other Properties as appropriate.
Set Properties("Roles")=roles
```

Your code could use different logic to obtain the information needed to define the user. You could instead obtain this information in the following ways:

- From IDToken if you are using OpenID Connect. In this case, call the **ValidateIDToken()** of `%SYS.OAuth2.Validate`.
- From Userinfo endpoint if you are OpenID Connect. In this case, call the **GetUserinfo()** of `%SYS.OAuth2.AccessToken`.

In any case, it is necessary to define a user whose username does not match a normal InterSystems IRIS username.

Your routine would also need to set roles and other parts of the *Properties* array as needed for your application. See [Create Delegated \(User-Defined\) Authentication Code](#) and [Delegated Authentication](#).

Revoking Access Tokens

If the authorization server supports token revocation, you can revoke access tokens via the Management Portal or programmatically.

Revoking a User's Access Tokens

To revoke all the access tokens for a given user, do the following:

1. Select **System Administration > Security > OAuth 2.0 > Administration**.
2. Type the user ID into the field **Revoke tokens for user**.
3. Select **Revoke**.

To perform this task, you must be logged in as a user who has USE permission on the %Admin_Secure resource.

Revoking Access Tokens Programmatically

If it is necessary for the client to revoke an access token, use the **RevokeToken()** method of %SYS.OAuth2.AccessToken. Note that when the session holding a given token is deleted, the system automatically calls this method (if a revocation endpoint is specified).

```
ClassMethod RevokeToken(applicationName As %String, accessToken As %String) As %Status
```

The arguments are as follows:

- *applicationName* is the name of the client application.
- *accessToken* is the access token.

The request is authorized using the basic authorization HTTP header with the *client_id* and *client_secret* associated with *applicationName*.

For example:

```
set sc=##class(%SYS.OAuth2.AccessToken).RevokeToken("myclient",accessToken)
if $$$ISERR(sc) {
    //error handling here
}
```

Note that you cannot use this method if the server does not specify a revocation endpoint or if **Client secret** is not specified.

%SYS.OAuth2.AccessToken also provides the method **RemoveAccessToken()**, which removes the access token from the client but does not remove the token from the server.

Rotating Keys Used for JWTs

In most cases, you can cause the client to generate new public/private key pairs; this applies only to the RSA keys used for the asymmetric RS256, RS384, and RS512 algorithms. (The exception is if you specify **Source other than dynamic registration** as **X509 certificate**. In this case, it is not possible to generate new keys.)

Generating new public/private key pairs is known as *key rotation*; this process adds new private RSA keys and associated public RSA keys to the private and public [JWKSs](#).

When you perform key rotation on the client, the client uses the new private RSA keys to sign JWTs to be sent to the authorization server. Similarly, the client uses the new public RSA keys to encrypt JWTs to be sent to the authorization server. To decrypt JWTs received from the authorization server, the client uses the new RSA keys, and if that fails, uses the old RSA keys; thus the client can decrypt a JWT that was created using its old public RSA keys. Last, if the client cannot verify a signed JWT received from the authorization server, then if the client has the URL for the authorization

server public JWKS, the client obtains a new public JWKS and tries again to verify the signature. (Note that the client has a URL for the authorization server public JWKS if the client was registered dynamically or if the configuration specified the **JWKS from URL** option; otherwise, the client does not have this URL.)

To rotate keys for a given client configuration:

1. In the Management Portal, select **System Administration > Security > OAuth 2.0 > Client Configuration**.
2. Select the server description with which the client configuration is associated.
The system then displays all client configurations associated with that server description.
3. Select the configuration of the client whose keys you want to rotate.
4. Select the **Rotate Keys** button.

Note: The symmetric HS256, HS384, and HS512 algorithms always use the client secret as the symmetric key.

API for Key Rotation on the Client

To rotate keys programmatically on the client, call the **RotateKeys()** method of OAuth2.Client.

To obtain a new authorization server public JWKS, call the **UpdateJWKS()** method of OAuth2.ServerDefinition.

For details on these methods, see the class reference.

Getting a New Public JWKS from the Authorization Server

In most cases, the authorization server generates a public/private pair of **JWKSs**. There are different ways in which the client can receive the public JWKS. One way is for the authorization server to provide the public JWKS at a URL; see the **JWKS from URL** option in [Manually Creating a Server Description \(No Discovery\)](#).

If the authorization server was defined with **JWKS from URL** and if the authorization server generates a new pair of JWKSs, you can cause the client to obtain the new public JWKS from the same URL. To do so:

1. In the Management Portal, select **System Administration > Security > OAuth 2.0 > Client Configuration**.
2. Select the server description with which the client configuration is associated.
The system then displays all client configurations associated with that server description.
3. Select the configuration of the client.
4. Select the **Update JWKS** button.

If the authorization server was not defined with **JWKS from URL** and if the authorization server generates a new pair of JWKSs, it is necessary to obtain the public JWKS, send it to the client, and load it from a file.

OAuth 2.0 Client Variations

The instructions on using an InterSystems IRIS® web application [as an OAuth 2.0 client](#) focus on the authorization code grant type.

In the [basic scenario](#), the client receives an access token from the authorization server and then optionally calls additional endpoints in the authorization server: the introspection endpoint, the Userinfo endpoint, or both. After that, the client calls the resource server. Notice that it is also possible for the resource server to independently call these endpoints.

This topic discusses some variations.

Disabling PKCE

By default, clients that use the authorization code grant type leverage the Proof Key for Code Exchange (PKCE) extension. In almost all cases, your client should not modify this default behavior because PKCE is an important and widely accepted security feature. However, in rare cases, you might want to disable PKCE.

To disable PKCE, modify the `properties` argument of `GetAuthorizationCodeEndpoint()` before calling the method. Your code should include the line:

```
Set properties("code_verifier")=""
```

For more information about the `properties` argument of `GetAuthorizationCodeEndpoint()`, see [Method Details](#).

Implicit Grant Type

In this variation, the client uses the implicit grant type when requesting tokens.

Configuration requirements: See the instructions in [Configuring a Client](#), but specify **Client Type** as appropriate for your use case.

Code requirements: The overall flow is similar to the one for the [authorization code grant type](#), but do not call `GetAuthorizationCodeEndpoint()`. Instead call the `GetImplicitEndpoint()` method of the `%SYS.OAuth2.Authorization` class:

```
ClassMethod GetImplicitEndpoint(applicationName As %String,
                               scope As %String,
                               redirectURL As %String,
                               idtokenOnly As %Boolean = 0,
                               responseMode As %String,
                               ByRef properties As %String,
                               Output isAuthorized As %Boolean,
                               Output sc As %Status
                               sessionId as %String="" ) As %String
```

The arguments are as follows:

- *applicationName* is the name of the client application.
- *scope* is a space-delimited list of scopes for which access is requested, for example: "openid profile scope3 scope4"

The default is determined by the [client configuration](#) for the given *applicationName*.

- *redirectURL* is the URL of the page to which the authorization server should redirect the browser after returning the access token to the client server.
- *idtokenOnly* enables you to obtain only an ID token. If this argument is 0, the method obtains both an access token and (if the request includes the appropriate scope) an ID token. If this argument is 1, the method does not obtain an access token.
- *responseMode* specifies the mode of the response from the authorization server. This can be "query" (the default), "fragment" or "form_post".
- *properties*, which is passed by reference, is a multidimensional array that contains any parameters to be added to the request. This array must have the following structure:

Array node	Array value
<i>properties(parametername)</i> where <i>parametername</i> is the name of a parameter	<p>Value of the given parameter. The value can be a scalar value, an instance of a dynamic object, or the UTF-8 encoded serialized form of a dynamic object.</p> <p>Use a dynamic object if you want the request to include a parameter whose value is a JSON object; a scenario is the <code>claims</code> parameter that is defined by OpenID Connect. For details on dynamic objects, see Using JSON.</p> <p>To use the <code>request</code> or <code>request_uri</code> parameter, see the section Passing Request Objects as JWTs.</p>

- *isAuthorized*, which is returned as output, equals 1 if there is a locally stored access token for this client and this session, and if that access token authorizes all the scopes given by the *scope* argument. This parameter equals 0 otherwise.
- *sc*, which is returned as output, contains the status code set by this method.
- *sessionId* specifies the session ID. Specify this only if you want to override the default session (`%session.SessionId`).

Also see [Variation: Performing the Redirect within OnPreHTTP](#).

Password Credentials Grant Type

In this variation, the client uses the password credentials grant type when requesting tokens. You can use this grant type when the client has the password belonging to the resource owner. The client application can simply perform an HTTP POST operation to the token endpoint, without any page redirection; InterSystems IRIS provides a method to do this.

Configuration requirements: See the instructions in [Configuring a Client](#), but note that you do not need to specify **Client Secret**. (In general, you should use the client secret only when the client secret is needed and it is possible to protect the client secret.)

Code requirements: Your application should do the following:

1. Call the **IsAuthorized()** method of `%SYS.OAuth2.AccessToken` and check the returned value (and possible error), as described in [Obtaining Tokens](#).
2. If **IsAuthorized()** returned 0, call the **GetAccessTokenPassword()** method of `%SYS.OAuth2.Authorization`.

```
ClassMethod GetAccessTokenPassword(applicationName As %String,
                                username As %String,
                                password As %String,
                                scope As %String,
                                ByRef properties As %String,
                                Output error As %OAuth2.Error) As %Status
```

The arguments are as follows:

- *applicationName* is the name of the client application.
- *username* is a username.
- *password* is the corresponding password.
- *scope* is a space-delimited list of scopes for which access is requested, for example: "scope1 scope2 scope3". The default is determined by the [client configuration](#) for the given *applicationName*.
- *properties*, which is passed by reference, is a multidimensional array that contains any parameters to be added to the request. This array must have the following structure:

Array node	Array value
<i>properties(parametername)</i> where <i>parametername</i> is the name of a parameter	Value of the given parameter. The value can be a scalar value, an instance of a dynamic object, or the UTF-8 encoded serialized form of a dynamic object. Use a dynamic object if you want the request to include a parameter whose value is a JSON object; a scenario is the <code>claims</code> parameter that is defined by OpenID Connect. For details on dynamic objects, see Using JSON .

- *error*, which is returned as output, is either null or is an instance of `OAuth2.Error` that contains error information.

This method performs an HTTP POST operation to the token endpoint, and then receives and saves the access token (if any).

3. Check the *error* argument and proceed accordingly.
4. Continue as described in [Examining the Token\(s\)](#) and [Adding an Access Token to an HTTP Request](#).

Client Credentials Grant Type

In this variation, the client uses the client credentials grant type when requesting tokens. This grant type enables the client application to communicate with the resource server independently from any user. There is no user context. The client application can simply perform an HTTP POST operation to the token endpoint, without any page redirection; InterSystems IRIS provides a method to do this.

Configuration requirements: See the instructions in [Configuring a Client](#). Make sure to specify the **Client Type** as **Confidential** and specify **Client Secret**.

Code requirements: Your application should do the following:

1. Call the **IsAuthorized()** method of `%SYS.OAuth2.AccessToken` and check the returned value (and possible error), as described in [Obtaining Tokens](#).
2. If **IsAuthorized()** returned 0, call the **GetAccessTokenClient()** method of `%SYS.OAuth2.Authorization`.

```
ClassMethod GetAccessTokenClient(applicationName As %String,
                                scope As %String,
                                ByRef properties As %String,
                                Output error As %OAuth2.Error) As %Status
```

The arguments are as follows:

- *applicationName* is the name of the client application.
- *scope* is a space-delimited list of scopes for which access is requested, for example: "scope1 scope2 scope3". The default is determined by the [client configuration](#) for the given *applicationName*.
- *properties*, which is passed by reference, is a multidimensional array that contains any parameters to be added to the request. See the *properties* argument for **GetAccessTokenPassword()**, in the [previous subsection](#).
- *error*, which is returned as output, is either null or is an instance of `OAuth2.Error` that contains error information.

This method performs an HTTP POST operation to the token endpoint, and then receives and saves the access token (if any).

3. Check the *error* argument and proceed accordingly.
4. Continue as described in [Examining the Token\(s\)](#) and [Adding an Access Token to an HTTP Request](#).

Performing the Redirect within OnPreHTTP

For the authorization code and implicit grant types, the [basic instructions](#) use the following steps:

1. Call the **IsAuthorized()** method of %SYS.OAuth2.AccessToken.
2. Call the **GetAuthorizationCodeEndpoint()** method (for the authorization code grant type) or call the **GetImplicitEndpoint()** method (for the implicit grant type).
3. Provide an option (such as a button) that opens the URL returned by the previous step, thus enabling the user to authorize the request

An alternative is to modify the **OnPreHttp()** method of the page class (in your application), so that it calls either the **GetAccessTokenAuthorizationCode()** method (for the authorization code grant type) or call the **GetAccessTokenImplicit()** method (for the implicit grant type). These methods cause the browser to navigate directly (if needed) to the authentication form of the authorization server, without first displaying any content of your page.

Passing Request Objects as JWTs

InterSystems IRIS also supports passing the request object as a JWT, as specified in [section 6](#) of the OpenID Connect Core specification. You can pass the request object [by value](#) or [by reference](#).

In both cases, you use methods of the %SYS.OAuth2.Request class. See the class reference for additional methods not described in this section.

Passing a Request Object by Value

To use the `request` parameter to pass the request object as a JWT:

1. Call the **MakeRequestJWT()** method of the %SYS.OAuth2.Request class:

```
ClassMethod MakeRequestJWT(applicationName As %String,
                          ByRef properties As %String,
                          Output sc As %Status) As %String
```

Where:

- *applicationName* is the name of the client application.
- *properties*, which is passed by reference, is a multidimensional array that contains any parameters to be added to the request. This array must have the following structure:

Array node	Array value
<i>properties(parametername)</i> where <i>parametername</i> is the name of a parameter	Value of the given parameter. The value can be a scalar value, an instance of a dynamic object, or the UTF-8 encoded serialized form of a dynamic object.

- *sc*, which is returned as output, contains the status code set by this method.

This method returns a string, which is the JWT. For example:

ObjectScript

```
// create jwt
set jwt=##class(%SYS.OAuth2.Request).MakeRequestJWT("myapp",.properties,.sc)
```

2. Modify the *properties* array that you will use as the argument for **GetAuthorizationCodeEndpoint()** or **GetImplicitEndpoint()**. Set the node *properties("request")* equal to the JWT that you created in the previous step. For example:

ObjectScript

```
set properties("request")=jwt
```

- When you call **GetAuthorizationCodeEndpoint()** or **GetImplicitEndpoint()**, include the *properties* array. For example:

ObjectScript

```
set url=##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint("myapp",
scope,redirect,.properties,.isAuthorized,.sc, responseMode)
```

Passing a Request Object by Reference

To use the `request_uri` parameter to pass the request object as a JWT:

- Call the **UpdateRequestObject()** method of the `%SYS.OAuth2.Request` class:

```
ClassMethod UpdateRequestObject(applicationName As %String,
requestName As %String,
ByRef properties As %String,
Output sc As %Status) As %SYS.OAuth2.Request
```

Where:

- applicationName* is the name of the client application.
- requestName* is the name of the request.
- properties*, which is passed by reference, is a multidimensional array that contains any parameters to be added to the request. This array must have the following structure:

Array node	Array value
<i>properties(parametername)</i> where <i>parametername</i> is the name of a parameter	Value of the given parameter. The value can be a scalar value, an instance of a dynamic object, or the UTF-8 encoded serialized form of a dynamic object.

- sc*, which is returned as output, contains the status code set by this method.

This method creates, saves, and returns an instance of `%SYS.OAuth2.Request`.

ObjectScript

```
// create requestobject
set
requestobject=##class(%SYS.OAuth2.Request).UpdateRequestObject("myapp","myrequest",.properties,.sc)
```

- Get the URL of the saved request object. To do so, call the **GetURL()** method of the instance. Note that **GetURL()** returns a status code as output in the first argument; your code should check that.

ObjectScript

```
Set requesturl=requestobject.GetURL()
```

- Modify the *properties* array that you will use as the argument for **GetAuthorizationCodeEndpoint()** or **GetImplicitEndpoint()**. Set the node *properties("request_uri")* equal to the URL obtained in the previous step. For example:

```
set properties("request_uri")=requesturl
```

4. When you call **GetAuthorizationCodeEndpoint()** or **GetImplicitEndpoint()**, include the *properties* array. For example:

```
set url=##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint("myapp",
scope,redirect,.properties,.isAuthorized,.sc, responseMode)
```

Calling Other Endpoints of the Authorization Server

The methods in %SYS.OAuth2.Authorization enable you to call a specific set of endpoints in the authorization server. If the authorization server has other endpoints, use the following general process to call them:

1. Create an instance of %Net.HttpRequest, set its properties as needed, and call methods as needed, in order to define the request.

```
Set httpRequest=##class(%Net.HttpRequest).%New()
Set httpRequest.ContentType="application/x-www-form-urlencoded"
...
```

For details on this class, see [Sending HTTP Requests](#) in *Using Internet Utilities*.

2. To add authentication to the request, call the **AddAuthentication()** method of %SYS.OAuth2.AccessToken.

```
ClassMethod AddAuthentication(applicationName As %String, httpRequest As %Net.HttpRequest) As %Status
```

Where:

- *applicationName* is the name of the OAuth 2.0 client.
- *httpRequest* is the instance of %Net.HttpRequest

InterSystems IRIS looks up the given client and uses its **Authentication type**, **SSL configuration**, and other information to add the appropriate authentication to the request.

3. Optionally open the client configuration so that you can use properties contained in it. To do so, switch to the %SYS namespace and call the **Open()** method of OAuth2.Client, passing the client name as the argument:

ObjectScript

```
New $NAMESPACE
set $NAMESPACE="%SYS"
Set client=##class(OAuth2.Client).Open(applicationName,.sc)
If client="" Quit
```

4. Call the **Post()**, **Get()**, or **Put()** method (as appropriate) of the HTTP request object, providing the authorization server's token endpoint as the argument. For example:

ObjectScript

```
set sc=httpRequest.Post(client.ServerDefinition.TokenEndpoint)
```

5. Perform additional processing as needed.

Using an InterSystems IRIS Web Application as an OAuth 2.0 Resource Server

This page describes how to use an InterSystems IRIS® web application as a resource server that uses the [OAuth 2.0 framework](#).

This page primarily discusses the scenario in which the resource server uses the introspection endpoint of the authorization server. See the [last section](#) for details on variations.

The process of [rotating keys](#) used for signing, encryption, signature verification, and decryption of JWTs is discussed elsewhere.

Prerequisites for the InterSystems IRIS Resource Server

Before starting the tasks described in this page, make sure the following items are available:

- An OAuth 2 authorization server.
- If the resource server uses any endpoint of the authorization server, the resource server may be registered as a client of the OAuth 2.0 authorization server. The details depend upon the implementation of the authorization server.

In this case, you will also later need to know specific details about this server:

- Location of the authorization server (issuer endpoint)
- Location of the token endpoint
- Location of the Userinfo endpoint (if supported; see [OpenID Connect Core](#))
- Location of the token introspection endpoint (if supported; see [RFC 7662](#))
- Location of the token revocation endpoint (if supported; see [RFC 7009](#))
- Whether the authorization server supports dynamic registration
- If the authorization server does not support dynamic registration, you will need the client ID and client secret for the resource server. The authorization server generates these two pieces of information (on a one-time basis) and you need to get them securely to the resource server machine.

Configuration Requirements

See [Configuration Requirements](#), with the following changes when you create the client configuration:

- For **Application name**, specify the application name of the resource server.
- For **Client Type**, specify **Resource Server**.

Note that when you specify **Resource Server** as the type, the configuration page displays only the options that are applicable to a resource server.

- For **clientId**, use the client ID of the resource server.
- For **clientSecret**, use the client secret of the resource server.

Code Requirements

An OAuth 2.0 resource server receives a request, examines the access token that it contains, and (depending on the access token) returns the requested information.

To create an InterSystems IRIS resource server, create a subclass of `%CSP.REST`, in the namespace used by the resource server's web application. In this class, create a [URL map](#) and the corresponding methods. In the methods, do the following:

1. Call the method **GetAccessTokenFromRequest()** of `%SYS.OAuth2.AccessToken`. This method is as follows:

```
ClassMethod GetAccessTokenFromRequest(Output sc As %Status) As %String
```

The method returns the access token, if any, found in the HTTP request received by this page. It uses one of the three [RFC 6750](#) formats. The parameter `sc`, returned as output, is a status code that indicates whether an error was detected. If the request did not use SSL/TLS, that is an error condition. Also, if the request did not include a valid bearer header, that is an error condition.

2. Check to see whether the status code is an error.

If the status is an error, the method should return a suitable error (and not return the requested information).

3. If the status code is not an error, validate the access token. To do so, use **ValidateJWT()** or your own custom method. See [Method Details](#).
4. Optionally call the **GetIntrospection()** method for additional information. This method calls the introspection endpoint of the authorization server and obtains claims about the access token. This method is as follows:

```
ClassMethod GetIntrospection(applicationName As %String,  
                             accessToken As %String,  
                             Output jsonObject As %RegisteredObject) As %Status
```

The arguments are as follows:

- *applicationName* is the name of the client application.
 - *accessToken* is the access token previously returned.
 - *jsonObject*, which is returned as output, is a JSON object that contains the claims that the authorization server makes about this access token.
5. If the preceding steps indicate that the user's request for information should be granted, perform the requested processing and return the requested information.

For example:

```
// This is a dummy resource server which just gets the access token from the request  
// and uses the introspection endpoint to ensure that the access token is valid.  
// Normally the response would not be security related, but would contain some interesting  
// data based on the request parameters.  
set accessToken=##class(%SYS.OAuth2.AccessToken).GetAccessTokenFromRequest(.sc)  
if $$$ISOK(sc) {  
    set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection("demo resource",accessToken,.jsonObject)  
  
    if $$$ISOK(sc) {  
        write "OAuth 2.0 access token used to authorize resource server (RFC 6749)<br>"  
        write "Access token validated using introspection endpoint (RFC 7662)<br>"  
        write "    scope='"_jsonObject.scope_"'<br>"  
        write "    user='"_jsonObject.username_"'<br>"  
    } else {  
        write "Introspection Error="_.EscapeHTML($system.Status.GetErrorText(sc)),!  
    }  
} else {  
    write "Error Getting Access Token="_.EscapeHTML($system.Status.GetErrorText(sc)),!  
}  
  
Quit $$$OK
```

Examining the Token(s)

After the resource server receives an access token, it should perform additional checks to determine whether the user has the necessary permissions to use the requested resource. To perform this examination, the client can use the methods described here to obtain additional information.

ValidateJWT()

Location: This method is in the class %SYS.OAuth2.Validation.

```
ClassMethod ValidateJWT(applicationName As %String,  
                        accessToken As %String,  
                        scope As %String,  
                        aud As %String,  
                        Output jsonObject As %RegisteredObject,  
                        Output securityParameters As %String,  
                        Output sc As %Status) As %Boolean
```

Use this method only if the access token is a JWT (rather than an opaque token).

This method decrypts the JWT if necessary, validates the JWT, and creates an object (*jsonObject*) to contain the JWT properties. To validate the JWT, the method checks the audience (if *aud* is specified), issuer endpoint (must match that specified in server description), and scope (if *scope* is specified). The method also makes sure the access token has not expired. Both signed and unsigned JWTs are accepted. If the JWT is signed, the method checks the signature.

This method returns 1 if the JWT is valid or returns 0 otherwise. It also returns several arguments as output.

The arguments are as follows:

- *applicationName* is the name of the client application.
- *accessToken* is the JWT to be validated.
- *scope* is a space-delimited list of scopes, for example: "scope1 scope2 scope3"

If *scope* is specified, the JWT must contain a scope claim that includes this scope.

- *aud* specifies the audience that is using the token. If the token has an associated *aud* property (usually because the audience was specified when requesting the token), then *aud* is matched to the token audience. If *aud* is not specified, then no audience checking takes place.
- *jsonObject*, which is returned as output, is a dynamic object that contains the claims in the JWT. This dynamic object contains properties such as *aud*, *exp*, *iss*, and so on. For details on dynamic objects, see [Using JSON](#).
- *securityParameters*, which is returned as output, is a multidimensional array that contains security information taken from the header, for optional additional use in verifying signatures, decrypting, or both.

This array contains the following nodes:

Node	Value
<i>securityParameters</i> ("sigalg")	The signature or MAC algorithm. Set only if the JWT is signed
<i>securityParameters</i> ("keyalg")	The key management algorithm
<i>securityParameters</i> ("encalg")	The content encryption algorithm

The *keyalg* and *encalg* nodes are either both specified or both null.

- *sc*, which is returned as output, contains the status code set by this method.

If this method returns success (1), examine *jsonObject*, and use the contained claims as needed to determine whether to allow access to the requested resource. Use *securityParameters* if needed.

Because the OAuth specification allows an application to accept both signed and unsigned JWTs, the `ValidateJWT` method does not reject an unsigned JWT. However, in many cases it is strongly recommended that your application implement stricter security by rejecting an unsigned JWT. You can determine whether the token passed into `ValidateJWT` was unsigned by inspecting the *securityParameters* array that is returned by the method. If *securityParameters*("sigalg") was not set, the token was unsigned. For example, the following code determines whether the token was unsigned and rejects it if it was:

```
Set tInitialValidationPassed = ##class(%SYS.OAuth2.Validation).ValidateJWT(tClientName,
tAccessToken, "", "", .tJsonObj,.tSecurityParams, .tValidateStatus)
// the "sigalg" subscript is set only if the JWT was signed
Set tIsTokenSigned = $Data(tSecurityParams("sigalg"))#2
If 'tIsTokenSigned {
    $$$ThrowStatus($System.Status.Error($$$AccessDenied))
}
```

GetIntrospection()

Location: This method is in the class %SYS.OAuth2.AccessToken.

```
ClassMethod GetIntrospection(applicationName As %String,  
                             accessToken As %String,  
                             Output jsonObject As %RegisteredObject) As %Status
```

This method sends the access token to the introspection endpoint, receives a response that contains claims, and creates an object (*jsonObject*) that contains the claims returned by that endpoint.

The request is authorized using the basic authorization HTTP header with the `client_id` and `client_secret` associated with *applicationName*.

The arguments are as follows:

- *applicationName* is the name of the client application.
- *accessToken* is the access token.
- *jsonObject*, which is returned as output, is a dynamic object that contains the claims returned by introspection endpoint. For details on dynamic objects, see [Using JSON](#).

Note that you cannot use this method if the server does not specify an introspection endpoint or if **Client secret** is not specified.

If this method returns success (1), examine *jsonObject*, and use the contained claims as needed to determine whether to allow access to the requested resource.

Variations

This page primarily discusses the scenario in which the InterSystems IRIS resource server uses the introspection endpoint of the authorization server. This section discusses some possible variations.

Variation: Resource Server Calls Userinfo Endpoint

The resource server can also call the Userinfo endpoint. To do so, the resource server code must use the **GetUserinfo()** method as you do for an [OAuth client](#).

Variation: Resource Server Does Not Call Endpoints

If the InterSystems IRIS resource server does not use any endpoints of the authorization server, it is not necessary to create an OAuth 2.0 configuration on this machine.

Also, the resource server does not need to use **GetAccessTokenFromRequest()**. Instead, it can get the access token directly from the HTTP authorization header and use it as needed.

Using InterSystems IRIS as an OAuth 2.0 Authorization Server

This page describes how to use an InterSystems IRIS® instance as an [OAuth 2.0](#) authorization server.

It is likely that the person who creates client definitions will not be the same person who set up the server. Moreover, it may be necessary to create client definitions on an ongoing basis. For this reason, the task of creating client definitions is included as a stand-alone section, at the end of the article.

Configuration Requirements for the InterSystems IRIS Authorization Server

To use an InterSystems IRIS instance as an OAuth 2.0 authorization server, perform the following configuration tasks:

- For the web server that is serving InterSystems IRIS, configure that web server to use SSL. It is beyond the scope of this documentation to describe how to configure a web server to use SSL.
- Create an InterSystems IRIS SSL configuration for use by the server.

This should be a client SSL configuration; no certificate is needed. The configuration is used to connect to a web server. Via this connection, the authorization server accesses the request object specified by the `request_uri` parameter. Via this connection, the authorization server also accesses the `jwtks_uri` when updating a client [JWKS](#). If the client does not send requests using the `request_uri` parameter and if the authorization server does not update the client JWKSs via the `jwtks_uri` parameter, then the authorization server does not need an SSL configuration.

For details on creating SSL configurations, see [InterSystems TLS Guide](#).

Each SSL configuration has a unique name. For reference, the documentation refers to this one as `sslconfig`, but you can use any unique name.

- Create the server configuration as described in the [following subsection](#).
- Later, create client definitions as needed; see the [last section](#).

Configuring the Authorization Server

In order to perform this task, you must be logged in as a user who has USE permission on the %Admin_Secure resource.

1. In the Management Portal, select **System Administration > Security > OAuth 2.0 > Server Configuration**.
2. On the **General** tab, specify the following details:
 - **Description** — Enter an optional description.
 - **Issuer endpoint** — Specify the endpoint URL of the authorization server. To specify this URL, enter values for the following options:
 - **Host name** — Specify the host name or IP address of the authorization server.
 - **Port** — Specify this if needed to accommodate any changes in the [Web Gateway](#) configuration.
 - **Prefix** — Specify this if needed to accommodate any changes in the [Web Gateway](#) configuration.

The resulting issuer endpoint has the following form:

```
https://hostname:port/prefix/oauth2
```

If you omit **Port**, the colon is omitted. Similarly, if you omit **Prefix**, there is only one slash between `hostname:port` and `oauth2`.

- **Audience required** — Specify whether the authorization server requires the `aud` parameter in authorization code and implicit requests. If this check box is clear, `aud` is not required.
- **Support user session** — Specify whether the authorization server supports user sessions (note that these are not web sessions). If selected, InterSystems IRIS uses a session maintenance class (the default session maintenance class is `OAuth2.Server.Session`, but this can be custom. See [Code Customization Options](#) for more details). If this check box is not selected, user sessions are not supported.
- **Allow public client refresh** — Specify whether to allow the server to process client refresh tokens. When you select this check box, the server does not require a client secret to process refresh tokens.
- **Enforce Proof Key for Code Exchange (PKCE) for public clients** — If your authorization server supports the authorization code grant type, selecting this check box will require public clients to provide a PKCE secret value when requesting the authorization code and exchanging the code for an access token.
- **Enforce Proof Key for Code Exchange (PKCE) for confidential clients** — If your authorization server supports the authorization code grant type, selecting this check box will require confidential clients to provide a PKCE secret value when requesting the authorization code and exchanging the code for an access token.

- **Support HTTP-based front channel logout** — Specify whether to enable or disable front channel logout. By default, front channel logout is enabled. If you clear this check box, then when users log out on the server, the server does also log users out of the clients that are registered with the server.

Note: For an InterSystems IRIS authorization server to support front channel logout, the **User Cookie Scope** for the `/oauth2` web application must be set to `Lax`. For details on configuring application settings, see [Create and Edit Applications](#).

- **Support sending sid (session ID) claim with front channel logout URL** — Specify whether to enable or disable sending the session ID claim along with the front channel logout URL. By default, sending the session ID is enabled. If you clear this check box, then when the server calls the front channel logout URL, it does not append the query parameters `iss` (issuer) and `sid` (session ID) to the URL.
- **Return refresh token** — Specify the conditions under which a refresh token is returned along with the access token. Select the option appropriate for your business case.
- **Supported grant types** — Specify the grant types that this authorization server allows to be used to create an access token. Select at least one.
- **OpenID provider documentation** — Specify URLs provided by the OpenID provider as follows:
 - **Service Documentation URL** — URL of a web page that provides human-readable information that developers might want or need to know when using the OpenID provider.
 - **Policy URL** — URL of a web page that describes the OpenID provider's policy on how a relying party can use the data provided by the provider.
 - **Terms of Service URL** — URL of a web page that describes the OpenID provider's terms of service.
- **SSL/TLS configuration** — Select the SSL/TLS configuration you created for use by the authorization server (for example, `sslconfig`).

3. On the **Scopes** tab, specify the following details:

- **Table with Scope and Description columns** — Specify all scopes supported by this authorization server.
- **Allow unsupported scope** — Specify whether the authorization server ignores scope values that it does not support. If this check box is clear, the authorization server returns an error when a request contains an unsupported scope value; in this case the request is not processed. If this check box is selected, the authorization server ignores the scope and processes the request.
- **Default scope** — Specify the default for access token scope if scope is not specified in the access token request or in the client configuration.

4. On the **Intervals** tab, specify the following details:

- **Access token interval** — Specify the number of seconds after which an access token issued by this server will expire. The default is 3600 seconds.
- **Authorization code interval** — Specify the number of seconds after which an authorization code issued by this server will expire. The default is 60 seconds.
- **Refresh token interval** — Specify the number of seconds after which a refresh token issued by this server will expire. The default is 24 hours (86400 seconds).
- **Session termination interval** — Specify the number of seconds after which a user session will be automatically terminated. The value 0 means the session will not be automatically terminated. The default is 24 hours (86400 seconds).

- **Client secret expiration interval** — Specify the number of seconds after which a client secret issued by this server will expire. The default value (0) means that the client secrets do not expire.

5. For the **JWT Settings** tab, specify the following details:

- **Create JWT Settings from X509 credentials** — Select this option if, for signing and encryption, you want to use the private key associated with a certificate; in this case, also see [Using Certificates for an OAuth 2.0 Authorization Server](#), in [Certificates and JWTs \(JSON Web Tokens\)](#).

Note: InterSystems expects that the option **Create JWT Settings from X509 credentials** will rarely be used, and that instead customers use the default behavior described next.

If you leave this option clear, the system generates a pair of **JWKSs** (JSON web key sets). One JWKS is private and contains all the needed private keys (per algorithm) as well as the client secret for use as a symmetric key; this JWKS is never shared. The other JWKS contains the corresponding public keys and is publicly available. InterSystems IRIS also copies the public JWKS to the client, so that the client can encrypt and verify signatures of JWTs from the authorization server.

- **Signing algorithm** — Select the algorithm to use when signing JWTs. Or leave this blank if JWTs are not to be signed.
- **Key management algorithm** — Select the algorithm to use for key management when encrypting JWTs.
Do this only if you select a content encryption algorithm.
- **Content encryption algorithm** — Select the algorithm to use when encrypting JWTs. Or leave this blank if JWTs are not to be encrypted. If you select an algorithm, you must also select an algorithm for key management.

6. On the **Customization** tab, specify details as described in [Code Customization Options](#).

7. Select **Save**.

When you save this configuration, the system creates a [web application](#) (/oauth2) for use by the authorization server. Do not modify this web application.

Code Customization Options and Overall Flow

This section describes the items in the **Customization Options** section of the [configuration options](#) of the authorization server. Subsections describe the [overall flow](#) and the [default classes](#).

- **Authenticate class** — Use %OAuth2.Server.Authenticate (the default) or a custom subclass of that class.
If you define a custom subclass, implement some or all of the following methods, depending on your needs:
 - **BeforeAuthenticate()** — Optionally implement this method to perform custom processing before authentication.
 - **DisplayLogin()** — Optionally implement this method to display a login page to identify the user. (For a less common alternative, see [Implementing DirectLogin\(\)](#).)
 - **DisplayPermissions()** — Optionally implement this method to display the requested permissions to the user.
 - **AfterAuthenticate()** — Optionally implement this method to perform custom processing after authentication.
- **Validate user class** — Use %OAuth2.Server.Validate (the default) or a custom class that defines the following methods:
 - **ValidateUser()** (used by all grant types other than client credentials)
 - **ValidateClient()** (used by the client credentials grant type)

InterSystems highly recommends that you define and use a custom class. The %OAuth2.Server.Validate class is provided for demonstration purposes and is very unlikely to be suitable for production use.

- **Session maintenance class** — The default session maintenance class, `OAuth2.Server.Session`, maintains user sessions via an HTTP-only cookie. This default class cannot be extended to implement custom logic. Rather, you can extend `OAuth2.Server.CookieSession` or `OAuth2.Server.AbstractSession`, depending on your requirements. The majority of the logic of the default class comes from `OAuth2.Server.CookieSession`, so if you want to implement a custom session maintenance class that takes advantage of existing code, including the use of an opaque browser cookie, extend `OAuth2.Server.CookieSession` for your custom class. Alternatively, you have the option of implementing an entirely custom session maintenance class that does not rely on default logic from InterSystems. To take this customization approach, extend `OAuth2.Server.AbstractSession` and implement the abstract methods.
- **Generate token class** — Use `OAuth2.Server.Generate` (the default), `OAuth2.Server.JWT`, or a custom class that defines the method **GenerateAccessToken()**. If you create a custom class, you might find it useful to subclass one of the classes listed here, because they provide methods you may want to use.
- **Customization namespace** — Specify the namespace in which the customization code should run.
- **Customization roles** — Specify the role or roles to use when running the customization code.

If you use any custom subclasses, see [Implementing the Custom Methods](#).

How an InterSystems IRIS Authorization Server Processes Requests

This section describes what an InterSystems IRIS authorization server does when it receives an authorization code request or an implicit request for a token.

1. Calls the **BeforeAuthenticate()** method of the class specified via the **Authenticate class** option. The purpose of this method is to make any modifications to the request before user identification starts.

In the [default class](#), this method is a stub.

2. Next, if the grant type is authorization code or implicit grant, InterSystems IRIS does the following:

- a. Calls the **DisplayLogin()** of the class specified via the **Authenticate class** option. (But also see [Implementing DirectLogin\(\)](#).)

In the [default class](#), **DisplayLogin()** displays a simple HTML login page.

- b. If the username is not null, calls the **ValidateUser()** method of the class specified via the **Validate user class** option. The purpose of this method is to validate the user and (by modifying the *properties* array) to prepare any claims to be returned by the token, Userinfo, and token introspection endpoints.

In the [default class](#), this method is only a sample and is very unlikely to be suitable for production use.

- c. If the user is validated, calls the **DisplayPermissions()** method of the class specified via the **Authenticate class** option. The purpose of this method is to display a page to the user that lists the requested permissions.

In the [default class](#), this method displays a simple HTML page with the permissions.

Or if the grant type is password credentials, InterSystems IRIS just calls the **ValidateUser()** method of the class specified via the **Validate user class** option.

Or if the grant type is client credentials, InterSystems IRIS just calls the **ValidateClient()** method of the class specified via the **Validate user class** option.

3. If the user accepts the permissions, calls the **AfterAuthenticate()** method of the class specified via the **Authenticate class** option. The purpose of this method is to perform any custom processing before generating an access token.

In the [default class](#), this method is a stub.

4. Calls the **GenerateAccessToken()** method of the class specified via the **Generate token class** option. The purpose of this method is to generate an access token to return to the user.

In the [default class](#) (%OAuth2.Server.Generate), this method generates an access token that is an opaque string. InterSystems IRIS also provides an alternative class (%OAuth2.Server.JWT), in which **GenerateAccessToken()** generates an access token that is a JWT.

Default Classes

This section describes the default classes in an InterSystems IRIS authorization server, as well as the class %OAuth2.Server.JWT, which is provided as another option for the **Generate token class**.

%OAuth2.Server.Authenticate (Default for Authenticate Class)

The class %OAuth2.Server.Authenticate defines the following methods, listed in the order in which they are called:

- **BeforeAuthenticate()** is a stub. It simply quits with an OK status.
- **DisplayLogin()** writes the HTML that creates a simple login page with **Login** and **Cancel** buttons.
- **DisplayPermissions()** writes the HTML that creates a simple page that displays the requested permissions. This page includes the buttons **Accept** and **Cancel**.
- **AfterAuthenticate()** is a stub. It simply quits with an OK status.

%OAuth2.Server.Validate (Default for Validate User Class)

The %OAuth2.Server.Validate class is the default class for the **Validate user class** option.

Note: This class is provided for sample purposes and is very unlikely to be suitable for production use. That is, InterSystems expects that customers will replace or subclass this class for their own needs.

This class defines the following sample methods:

- **ValidateUser()** does the following:
 1. Looks for the given user in the IRISYS database.
 2. Verifies the password for the user.
 3. Gets a multidimensional array that contains information about the user.
 4. Uses this array to add additional claims to the *properties* object.
- **SupportedClaims()** returns a \$LIST of claims that are supported by this authorization server. By default, this method specifically returns the list of claims defined by [OpenID Connect Core](#).
- **ValidateClient()** (used by the client credentials grant type) accepts all clients and adds no properties.

You can override all these methods in your subclass.

OAuth2.Server.Session (Default for Session Class)

The %OAuth2.Server.Session class is the default class for the **Session maintenance class** option. This class maintains sessions via an HTTP-only cookie.

In this class, the **GetUser()** method tries to access the current session. If there is a session, the method obtains the username from that session and returns that. If there is no session, the method returns the username as an empty string and also returns an error status as output.

For additional information on this class, see the class reference.

%OAuth2.Server.Generate (Default for Generate Token Class)

The %OAuth2.Server.Generate class is the default class for the **Generate token class** option. This class defines the following methods:

- **GenerateAccessToken()** generates a random string as the opaque access token.
- **IsJWT()** returns 0.

%OAuth2.Server.JWT (Another Option for Generate Access Token Class)

The %OAuth2.Server.JWT class is another class you can use (or subclass) for the **Generate token class** option. This class defines the following methods:

- **GenerateAccessToken()** returns a JWT. Before returning the JWT, InterSystems IRIS signs it, encrypts it, or both, according to the **JSON Web Token (JWT) Settings** in the [authorization server configuration](#).
- **IsJWT()** returns 1.
- **CreateJWT()** creates a JWT based on a JSON object containing the claims; signs and encrypts the JWT as specified in the [authorization server configuration](#). This method follows the specifications for OAuth 2.0 and OpenID Connect usage and should not be overridden in a subclass.
- **AddClaims()** — Adds the requested claims to the JWT. This method is as follows:

```
ClassMethod AddClaims(claims As %ArrayOfObjects,  
                     properties As %OAuth2.Server.Properties,  
                     json As %DynamicObject)
```

Where:

- *claims* is an array of %OAuth2.Server.Claim instances.
- *properties* is an instance of %OAuth2.Server.Properties that contains properties and claims that are used by the authorization server.
- *json* is a dynamic object that represents the JWT. The method modifies this object.

Implementing the Custom Methods for the InterSystems IRIS Authorization Server

To customize the behavior of the authorization server, define classes as described in [Code Customization Options](#). Then use this section for information on defining methods in those classes, depending on the processing steps that you want to customize.

1. [Optional custom processing before authentication](#)
2. [Identifying the user](#)
3. [Validating the user and specifying claims](#)
4. [Optionally displaying permissions to the user](#)
5. [Optional custom processing after authentication](#)
6. [Generating the access token](#)

After these subsections, a [final subsection](#) describes how to validate the client, in the case when this server must support the client credentials grant type. The client credentials grant type does not use steps 2 – 4 of the preceding list.

Optional Custom Processing Before Authentication

The information here applies to all grant types.

To perform custom processing before authenticating the user, implement the **BeforeAuthenticate()** method of the [Authenticate class](#). This method has the following signature:

```
ClassMethod BeforeAuthenticate(scope As %ArrayOfDataTypes,
                             properties As %OAuth2.Server.Properties) As %Status
```

Where:

- *scope* is an instance of %ArrayOfDataTypes that contains the scopes contained in the original client request. The array keys are the scope values and the array values are the corresponding display forms of those values.
- *properties* is an instance of %OAuth2.Server.Properties that contains properties and claims that are used by the authorization server. See [Details for the %OAuth2.Server.Properties Object](#).

In your method, optionally modify either or both of these arguments, both of which are later passed to the methods used to [identify the user](#). The method must return a %Status.

Normally, there is no need to implement this method. However, one use case is to implement the `launch` and `launch/patient` scopes used by FHIR[®], where the scope needs to be adjusted to include a specific patient.

Identifying the User

The information here applies only to the authorization code and implicit grant types.

To identify the user, implement the **DisplayLogin()** method of the [Authenticate class](#). The **DisplayLogin()** method has the following signature:

```
ClassMethod DisplayLogin(authorizationCode As %String,
                        scope As %ArrayOfDataTypes,
                        properties As %OAuth2.Server.Properties,
                        loginCount As %Integer = 1) As %Status
```

Where:

- *authorizationCode*
- *scope* is an instance of %ArrayOfDataTypes that contains the scopes contained in the original client request, possibly modified by the **BeforeAuthenticate()** method. The array keys are the scope values and the array values are the corresponding display forms of the scope values.
- *properties* is an instance of %OAuth2.Server.Properties that contains properties and claims received by the authorization server and modified by methods earlier in the processing. See [Details for the %OAuth2.Server.Properties Object](#).
- *loginCount* is the integer count of which login attempt is taking place.

This method is responsible for writing the HTML to display the user login form. The login form must contain a Username field, a Password field, and an AuthorizationCode field (which should be hidden). The default **DisplayLogin()** method uses of the **InsertHiddenField()** method of %CSP.Page to add the AuthorizationCode hidden field.

Typically, the form also has buttons with the values `Login` and `Cancel`. These buttons should submit the form. If the user submits the form with the `Login` button, the method will accept the username and password. If the user submits the form with the `Cancel` button, the authorization process will terminate with an error return of `access_denied`.

In your implementation, you might choose to display permissions on the same page. In that case, your method would display the scopes and would use a button named `Accept` to submit the page.

The method must return a %Status.

Updating properties.CustomProperties

If the form contains elements with names that start `p_`, such elements receive special handling. After the **DisplayLogin()** method returns, InterSystems IRIS adds values of those elements to the *properties.CustomProperties* array, first removing

the `p_` prefix from the names. For example, if the form contains an element named `p_addme`, then InterSystems IRIS adds `addme` (and the value of the `p_addme` element) to the `properties.CustomProperties` array.

Your method can also directly set other properties of `properties` as needed.

Validating the User and Specifying Claims

The information here applies to all grant types other than the client credentials grant type. (For that grant type, see [Validating the Client](#).)

To validate the user and specify any claims to be returned by the token, `Userinfo`, and token introspection endpoints, define the **`ValidateUser()`** method of the [Validate user class](#). This method has the following signature:

```
ClassMethod ValidateUser(username As %String,  
                        password As %String,  
                        scope As %ArrayOfDataTypes,  
                        properties As %OAuth2.Server.Properties,  
                        Output sc As %Status) As %Boolean
```

Where:

- `username` is the username provided by the user.
- `password` is the password provided by the user. Note that if the user has already logged in, InterSystems IRIS calls this method with `password` as an empty string. This means that your method should detect when `password` is an empty string and not attempt to check the password in that case.
- `scope` is an instance of `%ArrayOfDataTypes` that contains the scopes contained in the original client request, possibly modified by the **`BeforeAuthenticate()`** method. The array keys are the scope values and the array values are the corresponding display forms of the scope values.
- `properties` is an instance of `%OAuth2.Server.Properties` that contains properties and claims received by the authorization server and modified by methods earlier in the processing. See [Details for the %OAuth2.Server.Properties Object](#).
- `sc` is the status code set by this method. Use this to communicate details of any errors.

Your method should do the following:

- Make sure that the password applies to the given username.
- Use the `scope` and `properties` arguments as needed for your business needs.
- Modify the `properties` object to specify any claim values, as needed, or to add new claims. For example:

```
// Setup claims for profile and email OpenID Connect scopes.  
Do properties.SetClaimValue("sub",username)  
Do properties.SetClaimValue("preferred_username",username)  
Do properties.SetClaimValue("email",email)  
Do properties.SetClaimValue("email_verified",0,"boolean")  
Do properties.SetClaimValue("name",fullname)
```

- In the case of any errors, set the `sc` variable.
- Return 1 if the user is considered valid; return 0 in all other cases.

Note that after the return from **`ValidateUser()`**, the authorization server automatically sets the following values in the `properties` object, if these values are missing:

- In `properties.ClaimValues`:
 - `iss` — URL of authorization server
 - `sub` — `client_id`
 - `exp` — expiration time in seconds since December 31, 1840

- In *properties.CustomProperties*:
 - `client_id` — `client_id` of the requesting client

Displaying Permissions

The information here applies only to the authorization code and implicit grant types.

To display permissions after validating the user, implement the **DisplayPermissions()** method of the [Authenticate class](#). This method has the following signature:

```
ClassMethod DisplayPermissions(authorizationCode As %String,
                              scopeArray As %ArrayOfDataTypes,
                              currentScopeArray As %ArrayOfDataTypes,
                              properties As %OAuth2.Server.Properties) As %Status
```

Where:

- *authorizationCode* is the authorization code.
- *scopeArray* represents the newly requested scopes, for which the user has not yet granted permission. This argument is an instance of `%ArrayOfDataTypes`.

The array keys are the scope values and the array values are the corresponding display forms of the scope values.

- *currentScopeArray* represents the scopes for which the user has previously granted permission. This argument is an instance of `%ArrayOfDataTypes`.

The array keys are the scope values and the array values are the corresponding display forms of the scope values.

- *properties* is an instance of `%OAuth2.Server.Properties` that contains properties and claims received by the authorization server and modified by methods earlier in the processing. See [Details for the %OAuth2.Server.Properties Object](#).

This form must have buttons with the values `Accept` and `Cancel`. These buttons should submit the form. If the user submits the form with the `Accept` button, the method should continue with authorization. If the user submits the form with the `Cancel` button, the authorization process should terminate.

Optional Custom Processing After Authentication

The information here applies to all grant types.

To perform custom processing after authentication, implement the **AfterAuthenticate()** method of the [Authenticate class](#). This method has the following signature:

```
ClassMethod AfterAuthenticate(scope As %ArrayOfDataTypes, properties As %OAuth2.Server.Properties) As %Status
```

Where:

- *scope* is an instance of `%ArrayOfDataTypes` that contains the scopes as set by the authorization request and all processing before this method was called. The array keys are the scope values and the array values are the corresponding display forms of the scope values.
- *properties* is an instance of `%OAuth2.Server.Properties` that contains properties and claims as set by the authorization request and all processing before this method was called. See [Details for the %OAuth2.Server.Properties Object](#).

In your method, optionally modify either or both of these arguments. In particular, you may want to may add properties to the authentication HTTP response; to do so add properties to *properties.ResponseProperties*.

Normally, there is no need to implement this method. However, one use case is to implement the launch and launch/patient scopes used by FHIR[®], where it is necessary to adjust the scope to include a specific patient.

Generating the Access Token

The information here applies to all grant types.

To generate access tokens, implement the **GenerateAccessToken()** method of the [Generate token class](#). This method has the following signature:

```
ClassMethod GenerateAccessToken(properties As %OAuth2.Server.Properties, Output sc As %Status) As %String
```

Where:

- *properties* is an instance of %OAuth2.Server.Properties that contains properties and claims received by the authorization server and modified by methods earlier in the processing. See [Details for the %OAuth2.Server.Properties Object](#).
- *sc*, which is returned as output, is the status code set by this method. Set this variable to communicate details of any errors.

The method should return the access token. The access token may be based on the *properties* argument. In your method, you might also want to add claims to the JSON response object. To do so, set the ResponseProperties array property of the *properties* object.

Validating the Client

The information here applies only to the client credentials type.

To validate the client credentials and specify any claims to be returned by the token, Userinfo, and token introspection endpoints, define the **ValidateClient()** method of the [Validate user class](#). This method has the following signature:

```
ClassMethod ValidateClient(clientId As %String,  
                           clientSecret As %String,  
                           scope As %ArrayOfDataTypes,  
                           Output properties As %OAuth2.Server.Properties,  
                           Output sc As %Status) As %Boolean
```

Where:

- *clientId* is the client ID.
- *clientSecret* is the client secret.
- *scope* is an instance of %ArrayOfDataTypes that contains the scopes contained in the original client request, possibly modified by the **BeforeAuthenticate()** method. The array keys are the scope values and the array values are the corresponding display forms of the scope values.
- *properties* is an instance of %OAuth2.Server.Properties that contains properties and claims received by the authorization server and modified by methods earlier in the processing. See [Details for the %OAuth2.Server.Properties Object](#).
- *sc* is the status code set by this method. Use this to communicate details of any errors.

Your method should do the following:

- Make sure that the client secret applies to the given client ID.
- Use the *scope* and *properties* arguments as needed for your business needs.
- Modify the *properties* object to specify any claim values, as needed. For example:

```
// Setup claims for profile and email OpenID Connect scopes.  
Do properties.SetClaimValue("sub",username)  
Do properties.SetClaimValue("preferred_username",username)  
Do properties.SetClaimValue("email",email)  
Do properties.SetClaimValue("email_verified",0,"boolean")  
Do properties.SetClaimValue("name",fullname)
```

- In the case of any errors, set the *sc* variable.

- Return 1 if the user is considered valid; return 0 in all other cases.

Note that after the return from **ValidateClient()**, the authorization server automatically sets the following values in the *properties* object, if these values are missing:

- In *properties.ClaimValues*:
 - iss — URL of authorization server
 - sub — `client_id`
 - exp — expiration time in seconds since December 31, 1840
- In *properties.CustomProperties*:
 - `client_id` — `client_id` of the requesting client

Details for the %OAuth2.Server.Properties Object

The methods described in the previous section use the argument *properties*, which is an instance of %OAuth2.Server.Properties. The %OAuth2.Server.Properties class is intended to hold information that needs to be passed from method to method within the authorization server code. This section describes the [basic properties](#) in this class, as well as the [properties related to claims](#). The class also has [methods](#) for working with claims; the last subsection describes them.

Basic Properties

The %OAuth2.Server.Properties class has the following basic properties, used to convey information for any internal processing of your custom code:

RequestProperties

Property RequestProperties as array of %String (MAXLEN="");

Contains the query parameters from the authorization request.

Because this property is an array, use the usual array interface to work with it. (The same comment applies to the other properties of this class.) For example, to get the value of a query parameter, use `RequestProperties.GetAt(paramname)`, where *paramname* is the name of the query parameter.

ResponseProperties

Property ResponseProperties as array of %String (MAXLEN="");

Contains any properties to be added to the JSON response object to a token request. Set this property as needed.

CustomProperties

Property CustomProperties as array of %String (MAXLEN="");

Contains any custom properties to be used to communicate between various pieces of customization code. See [Updating properties.CustomProperties](#).

ServerProperties

Property ServerProperties as array of %String (MAXLEN="");

Contains any properties that the authorization server chooses to share with the customization code. The `logo_uri`, `client_uri`, `policy_uri` and `tos_uri` client properties are shared in this way for use by the Authentication Class.

Properties Related to Claims

The `%OAuth2.Server.Properties` class contains the `IntrospectionClaims`, `IDTokenClaims`, `UserinfoClaims`, and `JWTClaims` properties, which carry information about required claims, specifically custom claims.

The class also contains the `ClaimValues` property, which carries the actual claim values. Your customization code should set the values of the claims (typically in the `ValidateUser` class).

The following list describes these properties:

IntrospectionClaims

```
Property IntrospectionClaims as array of %OAuth2.Server.Claim;
```

Specifies the claims to be returned by the Introspection endpoint (beyond the base required claims). The authorization server will return the `scope`, `client_id`, `username`, `token_type`, `exp`, `iat`, `nbfi`, `sub`, `aud`, `iss`, and `jti` claims even if they are not in this property.

In most cases, the value of this property can be an empty string; this property is included to support the `claims` request parameter (see [OpenID Connect Core](#) section 5.5 for details).

Formally, this property is an array in which the array key is the claim name (which matches the name in the `ClaimValues` property) and the array value is an instance of `%OAuth2.Server.Claim`. The `%OAuth2.Server.Claim` class has the following properties:

- **Essential**

```
property Essential as %Boolean [ InitialExpression = 0 ];
```

Specifies whether the claim is essential or voluntary. The value 1 means essential and the value 0 means voluntary.

- **Values**

```
property Values as list of %String(MAXLEN=2048);
```

Specifies the list of permissible values for this claim.

The value of the claims will usually be set by the `ValidateUser` class.

IDTokenClaims

```
Property IDTokenClaims as array of %OAuth2.Server.Claim;
```

Specifies the claims that the authorization server requires in the IDToken (beyond the base set of required claims). The authorization server requires the `iss`, `sub`, `exp`, `aud`, and `azp` claims even if these claims are not in this property.

This property is an array of objects; for details, see the entry for the `IntrospectionClaims` property.

In most cases, the value of this property can be an empty string; this property is included to support the `claims` request parameter (see [OpenID Connect Core](#) section 5.5 for details).

UserinfoClaims

```
Property UserinfoClaims as array of %OAuth2.Server.Claim;
```

Specifies the claims to be returned by the Userinfo endpoint (beyond the base required claims). The authorization server will return the `sub` claim even if that claim is not in this property.

In most cases, the value of this property can be an empty string; this property is provided to support section 5.5 of OpenID Connect Core.

This property is an array of objects; for details, see the entry for the IntrospectionClaims property.

The claims are defined based on the scope and request claims parameter. The value to be returned for the claim will have the same key in the ClaimValues property. The value of the claims will usually be set by the ValidateUser class.

JWTClaims

Property JWTClaims as array of %OAuth2.Server.Claim;

Specifies the claims that are needed for the JWT access token that is returned by the default JWT-based access token class (%OAuth2.Server.JWT) beyond the base set of required claims. The authorization server will return the iss, sub, exp, aud, and jti claims even if they are not in this property.

This property is an array of objects; for details, see the entry for the IntrospectionClaims property.

The claims are defined by the customization code. The value of the claims will usually be set by the ValidateUser class.

ClaimValues

property ClaimValues as array of %String(MAXLEN=1024);

Specifies the actual claim values and their types. To work with this property, use the methods in the [next section](#).

If you need to work with this property directly, note that this property is an array in which:

- The array key is the claim name.
- The array value has the form `$LISTBUILD(type,value)`, where *type* holds the type of the value, and *value* holds the actual value. The *type* can be "string", "boolean", "number", or "object". If *type* is "object", then *value* is a JSON object serialized as a string.

Note that *value* can be a \$LIST structure. In this case, when the claim value is serialized, it is serialized as a JSON array, in which each array item has the given *type*.

Methods for Working with Claims

The %OAuth2.Server.Properties class also provides instance methods that you can use to work with that simplify working with the ClaimValues property.

SetClaimValue()

Method SetClaimValue(name As %String, value As %String, type As %String = "string")

Updates the ClaimValues property by setting the value of the claim named by the *name* argument. The *type* argument indicates the type of the claim: "string" (the default), "boolean", "number", or "object". If *type* is "object", then *value* must be a JSON object serialized as a string.

Note that *value* can be a \$LIST structure. In this case, when the claim value is serialized, it is serialized as a JSON array, in which each array item has the given *type*.

RemoveClaimValue()

Method RemoveClaimValue(name As %String)

Updates the ClaimValues property by removing the claim named by the *name* argument.

GetClaimValue()

```
Method GetClaimValue(name As %String, output type) As %String
```

Examines the ClaimValues property and returns the value of the claim named by the *name* argument. The *type* argument, which is returned as output, indicates the type of the claim; see **SetClaimValue()**.

NextClaimValue()

```
Method NextClaimValue(name As %String) As %String
```

Returns the name of the next claim (in the ClaimValues property) after the given claim.

Locations of the Authorization Server Endpoints

When you use an InterSystems IRIS instance as an OAuth 2.0 authorization server, the URLs for the authorization endpoints are as follows:

Endpoint	URL
Issuer endpoint	<code>https://serveraddress/oauth2</code>
Authorization endpoint	<code>https://serveraddress/oauth2/authorize</code>
Token endpoint	<code>https://serveraddress/oauth2/token</code>
Userinfo endpoint	<code>https://serveraddress/oauth2/userinfo</code>
Token introspection endpoint	<code>https://serveraddress/oauth2/introspection</code>
Token revocation endpoint	<code>https://serveraddress/oauth2/revocation</code>

In all cases, *serveraddress* is the IP address or host name of the server on which the InterSystems IRIS instance is running.

Creating Client Definitions on an InterSystems IRIS OAuth 2.0 Authorization Server

This section describes how to create a client definition on an InterSystems IRIS OAuth 2.0 authorization server, if you have not registered the client dynamically. First, set up the InterSystems IRIS OAuth 2.0 authorization server as described earlier in this page. Then use the Management Portal to do the following:

1. Select **System Administration > Security > OAuth 2.0 > Server Configuration**.
2. Click the **Client Configurations** button to view the client descriptions. This table is initially empty.
3. On the **General** tab, specify the following details:
 - **Name** — Specify the unique name of this client.
 - **Description** — Specify an optional description.
 - **Client type** — Specify the type of this client. The choices are **public** (a public client, per [RFC 6749](#)), **confidential** (a confidential client, per RFC 6749), and **resource** (a resource server which is not also a client).
 - **Redirect URLs** — One or more expected redirect URLs for this client.
 - **Supported grant types** — Specify the grant types that this client can use to create an access token. Select at least one.
 - **Supported response types** — Select the OAuth 2.0 response_type values that the Client will restrict itself to using.
 - **Authentication type** — Select the type of authentication (as specified in [RFC 6749](#) or [OpenID Connect Core section 9](#)) to be used for HTTP requests to the authorization server. Select one of the following:
 - none

- **basic**
 - **form encoded body**
 - **client secret JWT**
 - **private key JWT**
- **Authentication signing algorithm** — Select the algorithm that must be used for signing the JWTs used to authenticate this client at the token endpoint (if the authentication type is **client secret JWT** or **private key JWT**). If you do not select an option, any algorithm supported by the OpenID provider and the relying party may be used.
4. If needed, select the **Client Credentials** tab and view the following details:
 - **Client ID** — Client ID as specified in RFC 6749. InterSystems IRIS generates this string.
 - **Client secret** — Client secret as specified in RFC 6749. InterSystems IRIS generates this string.
 5. On the **Client Information** tab, specify the following details:
 - **Launch URL** — Specify the URL used to launch this client. In some circumstances, this value can be used to identify the client and can be used as the value of the `aud` claim.
 - **Authorization display** section:
 - **Client name** — Specify the name of the client to be presented to the end user.
 - **Logo URL** — Specify a URL that references a logo for the client application. If you specify this option, the authorization server displays this image to the end user during approval. The value of this field must point to a valid image file.
 - **Client home page** — Specify the URL of the home page of the client. The value of this field must point to a valid web page. If you specify this option, the authorization server displays this URL to the end user in a followable fashion.
 - **Policy URL** — Specify the URL that the Relying Party Client provides to the end user to read about the how the profile data will be used. The value of this field must point to a valid web page.
 - **Terms of service URL** — Specify the URL that the Relying Party Client provides to the end user to read about the Relying Party's terms of service. The value of this field must point to a valid web page.
 - **Contact emails** — Comma-separated list of email addresses suitable for use in contacting those responsible for the client application.
 - **Default max age** — Specify the default maximum authentication age, in seconds. If you specify this option, the end user must be actively re-authenticated when the maximum authentication age is reached. The `max_age` request parameter overrides this default value. If you omit this option, there is no default maximum authentication age.
 - **Default scope** — Specify the default scope, as a blank separated list, for access token requests.
 6. On the **JWT Settings** tab, specify the following details:
 - **JSON Web Token (JWT) Settings** — Specifies the source of the public keys that the client uses to verify signatures of JWTs from the authorization server and to encrypt JWTs sent to the authorization server.

By default, the dynamic registration process generates a pair of **JWKSs** (JSON web key sets). One JWKS is private and contains all the needed private keys (per algorithm) as well as the client secret for use as a symmetric key; this JWKS is never shared. The other JWKS contains the corresponding public keys and is publicly available. The dynamic registration process also copies the public JWKS to the client.

The other options are as follows:

- **JWKS from URL** — Specify a URL that points to a public JWKS and then load the JWKS into InterSystems IRIS.
- **JWKS from file** — Select a file that contains a public JWKS and then load that file into InterSystems IRIS.
- **X509 certificate** — For details, see [Using Certificates for an OAuth 2.0 Authorization Server](#), in [Certificates and JWTs \(JSON Web Tokens\)](#).

To access any of these options, first select **Source other than dynamic registration**.

7. Select **Save**.

Rotating Keys Used for JWTs

In most cases, you can cause the authorization server to generate new public/private key pairs; this applies only to the RSA keys used for the asymmetric RS256, RS384, and RS512 algorithms. (The exception is if you specify **Source other than dynamic registration** as **X509 certificate**. In this case, it is not possible to generate new keys.)

Generating new public/private key pairs is known as *key rotation*; this process adds new private RSA keys and associated public RSA keys to the private and public [JWKSs](#).

When you perform key rotation on the authorization server, the authorization server uses the new private RSA keys to sign JWTs to be sent to the clients. Similarly, the authorization server uses the new public RSA keys to encrypt JWTs to be sent to the clients. To decrypt JWTs received from the clients, the authorization server uses the new RSA keys, and if that fails, uses the old RSA keys; thus the server can decrypt a JWT that was created using its old public RSA keys.

Last, if the authorization server cannot verify a signed JWT received from a client, then if the authorization server has the URL for the client public JWKS, the authorization server obtains a new public JWKS and tries again to verify the signature. (Note that the authorization server has a URL for the client public JWKS if you used dynamic discovery or if the configuration specified the **JWKS from URL** option; otherwise, the authorization server does not have this URL.)

To rotate keys for the authorization server:

1. Select **System Administration > Security > OAuth 2.0 > Server Configuration**.

The system displays the configuration for the authorization server.

2. Select the **Rotate Keys** button.

Note: The symmetric HS256, HS384, and HS512 algorithms always use the client secret as the symmetric key.

API for Key Rotation on the Authorization Server

To rotate keys programmatically on the authorization server, call the **RotateKeys()** method of `OAuth2.Server.Configuration`.

To obtain a new client [JWKS](#), call the **UpdateJWKS()** method of `OAuth2.Server.Client`.

For details on these methods, see the class reference.

Getting a New Public JWKS from a Client

In most cases, a client generates a public/private pair of [JWKSs](#). There are different ways in which the authorization server can receive the public JWKS. One way is for the client to provide the public JWKS at a URL; see the **JWKS from URL** option in [Creating Client Definitions on an InterSystems IRIS OAuth 2.0 Authorization Server](#).

If the client was defined with **JWKS from URL** and if the client generates a new pair of JWKSs, you can cause the authorization server to obtain the new public JWKS from the same URL. To do so:

1. In the Management Portal, select **System Administration > Security > OAuth 2.0 > Server Configuration**.

The system displays the configuration for the authorization server.

2. Select the **Update JWKS** button.

If the client was not defined with **JWKS from URL** and if the client generates a new pair of JWKSs, it is necessary to obtain the public JWKS, send it to the authorization server, and load it from a file.

Creating Configuration Items Programmatically

Other articles describe how to use the Management Portal to configure OAuth 2.0 [clients](#), [resource servers](#), and [authorization servers](#). You can also create these configuration items programmatically. The following subsections provide the details for [clients](#) (including resource servers) and for the [authorization server](#).

Creating the Client Configuration Items Programmatically

To programmatically create the configuration items for an [OAuth 2.0 client](#) or an OAuth 2.0 resource server:

1. Create a [server description](#).
2. Create an associated [client configuration](#).

Creating a Server Description

A server description is an instance of `OAuth2.ServerDefinition`. To create a server description:

1. Switch to the `%SYS` namespace.
2. If the authorization server supports discovery, call the **Discover()** method of `%SYS.OAuth2.Registration`. This method is as follows:

```
ClassMethod Discover(issuerEndpoint As %String,
                    sslConfiguration As %String,
                    Output server As OAuth2.ServerDefinition) As %Status
```

Where:

- *issuerEndpoint* specifies the endpoint URL to be used to identify the authorization server.
- *sslConfiguration* specifies the alias of the InterSystems IRIS SSL/TLS configuration to use calling the **Discover()** method.
- *server*, which is returned as output, is an instance of `OAuth2.ServerDefinition`,

3. Then save the returned instance of `OAuth2.ServerDefinition`.

Or, if the authorization server does not support discovery:

1. Switch to the `%SYS` namespace.
2. Create an instance of `OAuth2.ServerDefinition`.
3. Set its properties. In most cases, the names of the properties match the labels shown in the Management Portal (apart from spaces and capitalization). For reference, see [Manually Creating a Server Description](#). The properties are as follows:
 - `IssuerEndpoint`
 - `SSLConfiguration`
 - `InitialAccessToken`, which corresponds to the **Registration access token** field.
 - `Metadata`, which is an instance of `OAuth2.Server.Metadata`, and which includes many properties. See OpenID Provider Metadata in https://openid.net/specs/openid-connect-discovery-1_0.html.

For information on `ServerCredentials`, see [Using Certificates for an OAuth 2.0 Authorization Server](#), in [Certificates and JWTs \(JSON Web Tokens\)](#).

4. Save the instance.

Creating a Client Configuration

A client configuration is an instance of `OAuth2.Client`. To create a client configuration:

1. Switch to the `%SYS` namespace.
2. Create an instance of `OAuth2.Client`.
3. Set its properties. In most cases, the names of the properties match the labels shown in the Management Portal (apart from spaces and capitalization). For reference, see [Configuring and Dynamically Registering a Client](#). The properties are as follows:
 - `ApplicationName`
 - `ClientId`, which you do not need to set manually if you will register the client dynamically.
 - `ClientSecret`, which you do not need to set manually if you will register the client dynamically.
 - `DefaultScope`
 - `Description`
 - `Enabled`
 - `JWTInterval`
 - `Metadata`, which is an instance of **OAuth2.Client.Metadata**, and which includes many properties. For information, see Client Metadata in http://openid.net/specs/openid-connect-registration-1_0-19.html.
 - `RedirectionEndpoint`, which corresponds to the option **The client URL to be specified to the authorization server to receive responses**. This property is of type `%OAuth2.Endpoint`. The class `OAuth2.Endpoint` is a serial class with the properties `UseSSL`, `Host`, `Port`, and `Prefix`.
 - `SSLConfiguration`
 - `ServerDefinition`, which must be an instance of `OAuth2.ServerDefinition` that you [created previously](#).

For information on `ClientCredentials`, see [Using Certificates for an OAuth 2.0 Client](#), in [Certificates and JWTs \(JSON Web Tokens\)](#).

4. If the authorization server supports dynamic client registration, call the **RegisterClient()** method of `%SYS.OAuth2.Registration`. This method is as follows:

```
ClassMethod RegisterClient(applicationName As %String) As %Status
```

Where *applicationName* is the name of the client application.

This method registers the client, retrieves client metadata (including the client ID and client secret), and then updates the instance of `OAuth2.Client`.

Creating the Server Configuration Items Programmatically

To programmatically create the configuration items for an [OAuth 2.0 authorization server](#):

1. Create an [authorization server configuration](#).

Note that you cannot define more than one authorization server configuration on any given InterSystems IRIS instance. Also, to create this configuration, you must be logged in as a user who has `USE` permission on the `%Admin_Secure` resource.

2. Create the associated [client descriptions](#).

Creating the Authorization Server Configuration

An authorization server configuration is an instance of `OAuth2.Server.Configuration`. To create an authorization server configuration:

1. Switch to the `%SYS` namespace.
2. Create an instance of `OAuth2.Server.Configuration`
3. Set its properties. In most cases, the names of the properties match the labels shown in the Management Portal (apart from spaces and capitalization). For reference, see [Configuring the Authorization Server](#). The properties are as follows:
 - `AccessTokenInterval`
 - `AllowUnsupportedScope`
 - `AudRequired`, which corresponds to the **Audience required** option
 - `AuthenticateClass`
 - `AuthorizationCodeInterval`
 - `ClientSecretInterval`
 - `CustomizationNamespace`
 - `CustomizationRoles`
 - `DefaultScope`
 - `Description`
 - `EncryptionAlgorithm`
 - `GenerateTokenClass`
 - `IssuerEndpoint`, which corresponds to the **Issuer endpoint** option, is of type `OAuth2.Endpoint`. The class `OAuth2.Endpoint` is a serial class with the properties `UseSSL`, `Host`, `Port`, and `Prefix`.
 - `JWKSFromCredentials`
 - `KeyAlgorithm`
 - `Metadata`, which is an instance of `OAuth2.Server.Metadata`, and which includes many properties. See OpenID Provider Metadata in https://openid.net/specs/openid-connect-discovery-1_0.html.
 - `RefreshTokenInterval`
 - `ReturnRefreshToken`
 - `SSLConfiguration`
 - `SessionClass`
 - `SessionInterval`, which corresponds to the **Session termination interval** option
 - `SigningAlgorithm`
 - `SupportSession`, which corresponds to the **Support user session** option
 - `SupportedScopes`, which corresponds to the table with **Scope** and **Description** columns. This property is an array of strings, and thus uses the usual array interface: `SetAt()`, `GetAt()`, and so on.
 - `ValidateUserClass`

For allowed values for algorithms for signing, key management, and encryption, the class reference for `%OAuth2.JWT`.

For information on `ServerCredentials` and `ServerPassword`, see [Using Certificates for an OAuth 2.0 Authorization Server](#), in [Certificates and JWTs \(JSON Web Tokens\)](#).

4. Save the instance using the `OAuth2.Server.Configuration.Save()` method. The `Save()` method should be used instead of the `%Save()` method because it provides additional functionality like creating a web application.

Note that InterSystems IRIS does not support having more than one instance of this class.

Also note that in order to save this instance, you must be logged in as a user who has `USE` permission on the `%Admin_Secure` resource.

Creating a Client Description

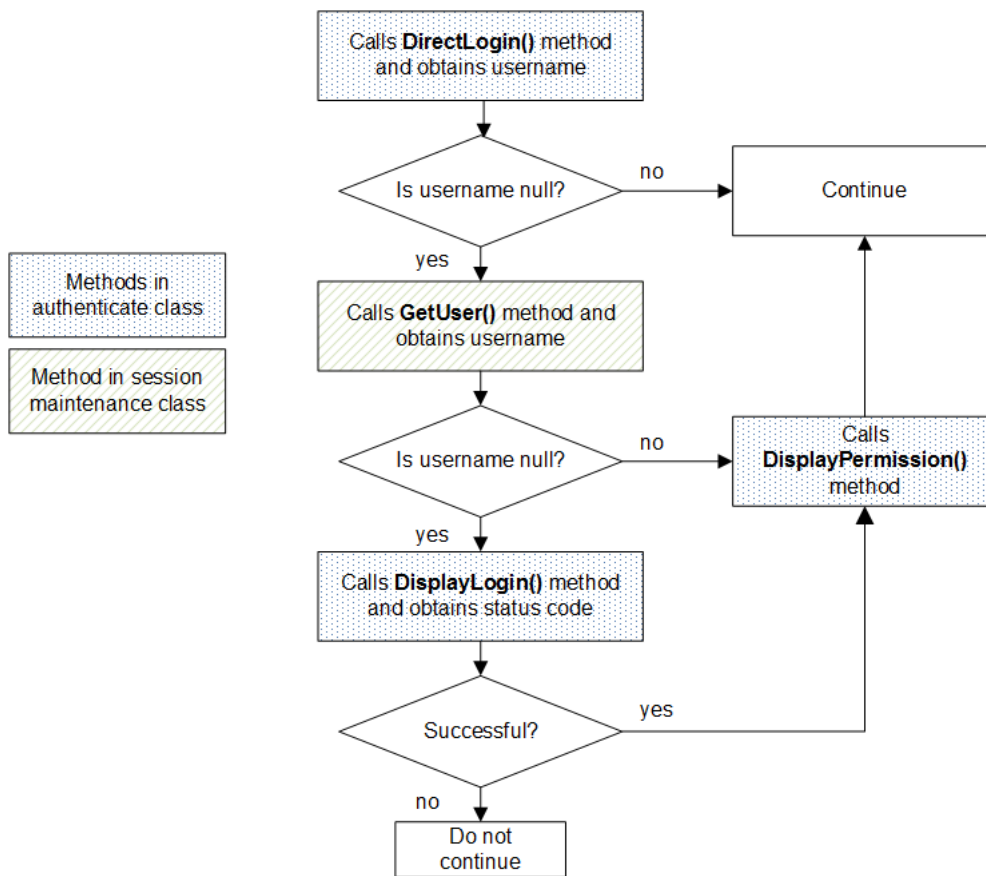
A client description is an instance of `OAuth2.Server.Client`. To create a client description:

1. Switch to the `%SYS` namespace.
2. Create an instance of `OAuth2.Server.Client`.
3. Set its properties. In most cases, the names of the properties match the labels shown in the Management Portal (apart from spaces and capitalization). For reference, see [Creating a Client Description](#). The properties are as follows:
 - `ClientCredentials`
 - `ClientType`
 - `DefaultScope`
 - `Description`
 - `LaunchURL`
 - `Metadata`, which is an instance of `OAuth2.Client.Metadata`, and which includes many properties. For information, see Client Metadata in http://openid.net/specs/openid-connect-registration-1_0-19.html.
 - `Name`
 - `RedirectURL`, which corresponds to the **Redirect URLs** option. This property is an array of strings, and thus uses the usual array interface: `SetAt()`, `GetAt()`, and so on.
4. Save the instance.

The system generates values for the `ClientId` and `ClientSecret` properties.

Implementing DirectLogin()

When you use InterSystems IRIS® as an OAuth 2.0 [authorization server](#), normally you implement the **DisplayLogin()** method of the [Authenticate class](#), which displays a page where the user enters a username and password and logs in. If you instead want the server to authenticate without displaying a login form and without using the current session, then implement the **DirectLogin()** method of the [Authenticate class](#). The following flowchart shows how an InterSystems IRIS authorization server identifies the user, when processing a request for an access token:



By default, the **GetUser()** method gets the username that was entered in the previous login.

Note that **DisplayPermissions()** is not called if you implement **DirectLogin()**, because **DirectLogin()** takes responsibility for displaying permissions.

The **DirectLogin()** method has the following signature:

```

ClassMethod DirectLogin(scope As %ArrayOfDataTypes,
    properties As %OAuth2.Server.Properties,
    Output username As %String,
    Output password As %String) As %Status
    
```

Where:

- scope* is an instance of %ArrayOfDataTypes that contains the scopes contained in the original client request, possibly modified by the **BeforeAuthenticate()** method. The array keys are the scope values and the array values are the corresponding display forms of the scope values.
- properties* is an instance of %OAuth2.Server.Properties that contains properties and claims received by the authorization server and modified by methods earlier in the processing. See [Details for the %OAuth2.Server.Properties Object](#).
- username*, returned as output, is a username.
- password*, returned as output, is the corresponding password.

In your implementation, use your own logic to set the *username* and *password* arguments. To do so, use the *scope* and *properties* arguments as needed. To deny access, your method can set the *username* argument to \$char(0). In this case, the authorization server will return the `access_denied` error.

The method can also set properties of *properties*; this object is available in later processing.

The method must return a %Status.

Certificates and JWTs (JSON Web Tokens)

Each party in [OAuth 2.0](#) requires public/private key pairs. For these pairs, you can use certificates and their private keys, although this is not the typical technique. This page provides the details for each of the following scenarios.

- [OAuth 2.0 client](#)
- [OAuth 2.0 resource server](#)
- [OAuth 2.0 authorization server](#)

In each case, to generate the private keys and corresponding certificates, you can use the [InterSystems public key infrastructure](#).

Note: InterSystems IRIS can generate a pair of JWKSs (JSON web key sets). One JWKS is private and contains all the needed private keys (per algorithm) as well as the client secret for use as a symmetric key; this JWKS is never shared. The other JWKS contains the corresponding public keys and is publicly available. If you want to use the option of generating JWKSs, ignore this page.

Using Certificates for an OAuth 2.0 Client

An [OAuth 2.0 client](#) can receive JWTs (which might be encrypted, signed, or both) from the authorization server. Similarly, the client can send JWTs (which might be encrypted, signed, or both) to the authorization server. If you would like to use certificate/private key pairs for these purposes, consult the table below to determine which certificates you need:

Scenario	Requirement for Client Configuration
Either: <ul style="list-style-type: none"> • Client needs to verify signatures of JWTs sent by authorization server • Client needs to encrypt JWTs sent to authorization server 	Obtain a certificate owned by the authorization server, as well as the CA (certificate authority) certificate that signs the server certificate. The public key in this certificate is used for signature verification and encryption.
Either: <ul style="list-style-type: none"> • Client needs to sign JWTs before sending to authorization server • Client needs to decrypt JWTs sent by authorization server 	Obtain a private key for the client, as well as the corresponding certificate and the CA certificate that signs the certificate. The private key is used for signing and decryption.

In each case, it is also necessary to do the following on the same instance that contains the client web application:

- Provide [trusted certificates](#) for InterSystems IRIS to use. The trusted certificates must include the certificates that sign the client's certificate and the authorization server's certificate (either or both, depending on which certificates you need).
- Create an InterSystems IRIS [credential set](#) that enables InterSystems IRIS to use the certificate.

For the client certificate, when you create the credential set, be sure to load the private key and provide the password for the private key.

- When you [configure the client](#), select the option **Create JWT Settings from X509 credentials**. Also specify the following:
 - **X509 credentials** — Select the credential set that uses the client's certificate and that contains the corresponding private key (for example, `ClientConfig`).

- **Private key password** — Enter the password for the private key for this certificate.

Using Certificates for an OAuth 2.0 Resource Server

An OAuth 2.0 [resource server](#) can receive JWTs (which might be encrypted, signed, or both) from the authorization server. Similarly, the resource server can send JWTs (which might be encrypted, signed, or both) to the authorization server. If you would like to use certificate/private key pairs for these purposes, consult the table below to determine which certificates you need:

Scenario	Requirement for Resource Server Configuration
Resource server needs to verify signatures of JWTs sent by authorization server	Obtain a certificate owned by the authorization server, as well as the CA certificate that signs the server certificate. The public key in this certificate is used for signature verification and encryption.
Resource server needs to encrypt JWTs sent to authorization server	
Resource server needs to sign JWTs before sending to authorization server	Obtain a private key for the resource server, as well as the corresponding certificate and the CA certificate that signs the certificate. The private key is used for signing and decryption.
Resource server needs to decrypt JWTs sent by authorization server	

In each case, it is also necessary to do the following on the same instance that contains the resource server web application:

- Provide [trusted certificates](#) for InterSystems IRIS to use. The trusted certificates must include the certificates that sign the resource server's certificate and the authorization server's certificate (either or both, depending on which certificates you need).
- Create an InterSystems IRIS [credential set](#) that enables InterSystems IRIS to use the certificate.
For the resource server's certificate, when you create the credential set, be sure to load the private key and provide the password for the private key.
- When you [configure the resource server](#), select the option **Create JWT Settings from X509 credentials**. Also specify the following:
 - **X509 credentials** — Select the credential set that uses the resource server's certificate and that contains the corresponding private key (for example, ResourceConfig).
 - **Private key password** — Enter the password for the private key for this certificate.

Using Certificates for an OAuth 2.0 Authorization Server

An OAuth 2.0 [authorization server](#) can receive JWTs (which might be encrypted, signed, or both) from its clients. Similarly, it can send JWTs (which might be encrypted, signed, or both) to its clients. If you would like to use certificate/private key pairs for these purposes, consult the table below to determine which certificates you need:

Scenario	Requirement for Authorization Server Configuration
Authorization server needs to verify signatures of JWTs sent by a client	Obtain a certificate owned by that client, as well as the CA certificate that signs the certificate. The public key in this certificate is used for signature verification and encryption.
Authorization server needs to encrypt JWTs sent to a client	
Authorization server needs to sign JWTs before sending to its clients	Obtain a private key for the authorization server, as well as the corresponding certificate and the CA certificate that signs the certificate. The private key is used for signing and decryption.
Authorization server needs to decrypt JWTs sent by its clients	

In each case, it is also necessary to do the following on the same instance that contains the authorization server:

- Provide [trusted certificates](#) for InterSystems IRIS to use. The trusted certificates must include the certificates that sign the clients' certificates and the authorization server's certificate (either or both, depending on which certificates you need).
- Create an InterSystems IRIS [credential set](#) that enables InterSystems IRIS to use the certificate.
For the authorization server certificate, when you create the credential set, be sure to load the private key and provide the password for the private key.
- When you [configure the server](#), select the **JWT Settings** tab. On that tab, select the option **Create JWT Settings from X509 credentials**. Also specify the following:
 - **X509 credentials** — Select the credential set that uses the authorization server's certificate and that contains the corresponding private key (for example, `AuthConfig`).
 - **Private key password** — Enter the password for the private key for this certificate.
- When you [create client definitions on the server](#), select the **JWT Settings** tab. On that tab, for **Source other than dynamic registration**, select **X509 certificate**. Also, for **Client credentials** — Select the credential set that uses the client's certificate (for example, `ClientConfig`).

Working with JWT Headers

This topic discusses how to customize the header of a JWT and how to process custom values in a JWT header. Keep in mind that a JWT can be generated by the authorization server or by custom code outside of the OAuth framework.

Adding Header Values (Authorization Server)

When the authorization server generates a JWT token, the `alg`, `enc`, `kid`, `typ` and `cty` headers are set automatically based on the signature and encryption algorithms used and cannot be directly manipulated with custom code. However, other header values can be added to the JWT header using the `JWTHeaderClaims` array. For example, `JWTHeaderClaims` can be used to add `jku` and `jwk` header parameters to the token. In the case of `jku`, the server automatically adds the relevant `jwk_uri` or `JKWS` to the JOSE array. Note that not all header parameters defined by RFC 7515 are supported. The `JWTHeaderClaims` array can also be used to add arbitrary custom values to the JWT header.

For example, the following code could be added to a subclass of %OAuth2.Server.Validate to add two standard headers and one custom header value to the JWT header:

```
ClassMethod ValidateUser(username As %String, password As %String, scope As %ArrayOfDataTypes, properties
As %OAuth2.Server.Properties, Output sc As %Status) As %Boolean
{
    ...
    Do properties.SetClaimValue("co","intersystems")

    Do properties.JWTHeaderClaims.SetAt("","jku")
    Do properties.JWTHeaderClaims.SetAt("","co")
    Do properties.JWTHeaderClaims.SetAt("","iss")
    ...
}
```

Adding Header Values (Direct JWT Generation)

It is possible to use custom code to generate a JWT outside of the OAuth framework using the `ObjectToJWT()` method. This method accepts an array of strings representing the JOSE header as its first parameter. To add values to the JWT header, simply add values to the JOSE array before calling the `ObjectToJWT` method. For example, to add the `jku` header parameter to the JOSE header:

```
Set JOSE("jku")=""
Set JOSE("jku_local")=%server.Metadata."jwks_uri"
Set JOSE("jku_remote")=%client.Metadata."jwks_uri"
// set JOSE("sigalg") and/or JOSE("encalg") and JOSE("keyalg")
Set sc=##class(%OAuth2.JWT).ObjectToJWT(.JOSE,json,%server.PrivateJWKS,%client.RemotePublicJWKS,.JWT)
```

For a description of JOSE array nodes that correspond to standard header parameters, see the class reference.

Adding Custom Header Parameters

The JOSE array passed to the `ObjectToJWT` method can include custom header parameters that are inserted into the JWT header. To include custom header parameters, first define them as key-value pairs in a [dynamic object](#), where the key is the name of the custom parameter and the value is the parameter's value. Once the dynamic object is defined, add it to a node of the JOSE array using the custom subscript. For example, the following code inserts two custom parameters, `co` and `prod`, into the JWT header:

```
Set newParams={"co":"intersystems","prod":"bazbar"}
Set JOSE("custom")=newParams
// set JOSE("sigalg") and/or JOSE("encalg") and JOSE("keyalg")
Set sc=##class(%OAuth2.JWT).ObjectToJWT(.JOSE,json,%server.PrivateJWKS,%client.RemotePublicJWKS,.JWT)
```

The custom node of the JOSE array cannot be used to override the header values defined by [RFC 7515](#). If doing nested signing and encryption, the custom headers will only be included in the "inner" (signed) token.

Processing Custom Header Parameters

The `JWTToObject` method allows you to process a JWT to return its headers. These headers can be accessed in two ways. Standard JOSE header parameters containing the algorithms used for Signature and/or Encryption operations performed on the JWT are returned by the method in an array of strings. In addition, the "raw" header of the JWT is returned as a [dynamic object](#) in the 6th parameter. By parsing the key-value pairs of this dynamic object, you can process custom and standard header parameters that are present in the JWT header. If the token was created using nested signing and encryption, the raw header returned by the method is from the "inner" (signed) token.

Delegated Authorization

Using Delegated Authorization

InterSystems IRIS® data platform supports the use of user-defined authorization code. This is known as *delegated authorization*.

- [Overview of Delegated Authorization](#)
- [Create Delegated \(User-Defined\) Authorization Code](#)
- [Configure an Instance to Use Delegated Authorization](#)
- [After Authorization — The State of the System](#)

Overview of Delegated Authorization

Delegated authorization allows administrators to implement custom mechanisms to replace the role-assignment activities that are part of InterSystems security. For example, user-defined authorization code might look up a user's roles in an external database and provide that information to InterSystems IRIS.

To use delegated authorization, there are the following steps:

1. [Create Delegated \(User-Defined\) Authorization Code](#) in the **ZAUTHORIZE** routine.
2. [Configuring an Instance to Use Delegated Authorization](#) for the InterSystems IRIS instance.

Note: Delegated authorization is only supported with Kerberos and Operating-System–based authentication.

Interactions between Delegated Authentication and Delegated Authorization

Delegated authorization through **ZAUTHORIZE.mac** is not supported for use with delegated authentication. The routine for delegated authentication (**ZAUTHENTICATE**, which is described in [Using Delegated Authentication](#)) provides support for authorization. When using **ZAUTHENTICATE**, you have the option to segregate authentication and authorization code.

Important: If using authentication with HealthShare®, you must use the [ZAUTHENTICATE routine provided by InterSystems](#) and cannot write your own.

Create Delegated (User-Defined) Authorization Code

Topics associated with creating delegated authorization code include:

- [Start From the ZAUTHORIZE.mac Template](#)
- [ZAUTHORIZE Signature](#)
- [Authorization Code with ZAUTHORIZE](#)
- [ZAUTHORIZE Return Value and Error Messages](#)

Start From the ZAUTHORIZE.mac Template

InterSystems provides a sample routine, **ZAUTHORIZE.mac**, that you can copy and modify. This routine is part of the Samples-Security sample on GitHub (<https://github.com/interSystems/Samples-Security>). You can download the entire sample as described in [Downloading Samples for Use with InterSystems IRIS](#), but it may be more convenient to simply open the file on GitHub and copy its contents.

To create your own **ZAUTHORIZE.mac**:

1. To use ZAUTHORIZE.mac as a template, copy its contents and save them into a ZAUTHORIZE.mac routine in the %SYS namespace.
2. Review the comments in the ZAUTHORIZE.mac sample. These provide important guidance about how to implement a custom version of the routine.
3. Edit your routine by adding custom authorization code and any desired code to set user account characteristics.

CAUTION: Because InterSystems IRIS places no constraints on the authorization code in **ZAUTHORIZE**, the application programmer is responsible for ensuring that this code is sufficiently secure.

Upgrade Delegated Authorization Code

Before upgrading to a new version of InterSystems IRIS, check ZAUTHORIZE.mac to determine if your current authorization code needs any changes to support new functionality.

ZAUTHORIZE Signature

When configured for delegated authorization, the system automatically calls **ZAUTHORIZE** after authentication occurs. InterSystems IRIS supplies values for the parameters defined in the **ZAUTHORIZE** signature as necessary. The signature of **ZAUTHORIZE** is:

ObjectScript

```
ZAUTHORIZE(ServiceName, Namespace, Username, Password,
           Credentials, Properties) PUBLIC {

    // authorization code
    // optional code to specify user account properties and roles
}
```

where:

- *ServiceName* — A string specifying the name of the service through which the user is connecting to InterSystems IRIS, such as `%Service_Console` or `%Service_WebGateway`.
- *Namespace* — A string specifying the namespace on the InterSystems IRIS server to which a connection is being established. This is for use only with `%Service_Bindings`, such as connections for Studio or ODBC; for any other service, the value should be `" "` (the empty quoted string).
- *Username* — A string specifying the user whose privileges are being determined.
- *Password* — A string specifying the password associated with account in use. This is for use only with the Kerberos K5API authentication mechanism; for any other mechanism, the value should be `" "` (the empty quoted string).
- *Credentials* — *Passed by reference*. Not implemented in this version of InterSystems IRIS.
- *Properties* — *Passed by reference*. An array of returned values that specifies characteristics of the account named by *Username*. For more information, see [ZAUTHORIZE and User Properties](#).

Authorization Code with ZAUTHORIZE

The purpose of **ZAUTHORIZE** is to establish or update the roles and other characteristics for the authenticated user. The content of authorization code is application-specific. It can include any user-written ObjectScript code, class method, or \$ZF callout.

ZAUTHORIZE specifies role information by setting the values of the *Properties* array, which is passed by reference to **ZAUTHORIZE**. Typically, the source for the values being set is a repository of user information that is available to **ZAUTHORIZE**.

CAUTION: Because InterSystems IRIS does not and cannot place any constraints on the authorization code in **ZAUTHORIZE**, the application programmer is responsible for ensuring that this code is sufficiently secure.

ZAUTHORIZE and User Properties

Elements of the *Properties* array specify values of attributes associated with the user specified by the *Username* parameter. Typically, code within **ZAUTHORIZE** sets values for these elements. The elements in the *Properties* array are:

- *Properties("Comment")* — Any text.
- *Properties("FullName")* — The first and last name of the user.
- *Properties("NameSpace")* — The default namespace for a Terminal login.
- *Properties("Roles")* — The comma-separated list of roles that the user holds in InterSystems IRIS.
- *Properties("Routine")* — The routine that is executed for a Terminal login. A value of " " specifies that the Terminal run in [programmer mode](#).
- *Properties("Password")* — The user's password.
- *Properties("Username")* — The user's username.

Each of these elements is described in more detail in one of the following sections.

Note: It is not possible to manipulate the value of any member of the *Properties* array after authorization.

Comment

If **ZAUTHORIZE** returns a value for *Properties("Comment")*, then that string becomes the value of the user account's *Comment* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) If no value is passed back to the calling routine, then the value of *Comment* for the user account is a null string and the relevant field in the Management Portal holds no content.

FullName

If **ZAUTHORIZE** returns a value for *Properties("FullName")*, then that string becomes the value of the user account's *Full name* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) If no value is passed back to the calling routine, then the value of *Full name* for the user account is a null string and the relevant field in the Management Portal holds no content.

NameSpace

If **ZAUTHORIZE** sets the value of *Properties("NameSpace")*, then that string becomes the value of the user account's *Startup NameSpace* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) If no value is passed back to the calling routine, then the value of *Startup NameSpace* for the user account is a null string and the relevant field in the Management Portal holds no content.

Once connected to InterSystems IRIS, the value of *Startup NameSpace* — as specified by the value of *Properties("NameSpace")* — determines the initial namespace for any user authenticated for local access (such as for Console, Terminal, or Telnet). If *Startup NameSpace* has no value, then the initial namespace for any user authenticated for local access is determined as follows:

1. If the USER namespace exists, that is the initial namespace.

2. If the USER namespace does not exist, the initial namespace is the %SYS namespace.

Note: If the user does not have the appropriate privileges for the initial namespace, access is denied.

Password

If **ZAUTHORIZE** sets the value of *Properties("Password")*, then that string becomes the value of the user account's *Password* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) If no value is passed back to the calling routine, then the value of *Password* for the user account is a null string and the relevant field in the Management Portal then holds no content.

If **ZAUTHORIZE** returns a password, this allows the user to log in to the system via Password authentication if it is enabled. This is a possible mechanism to help migrate from delegated authentication to Password authentication, though with the usual cautions associated with the use of multiple authentication mechanisms; see [Cascading Authentication](#) for more details.

Roles

If **ZAUTHORIZE** sets the value of *Properties("Roles")*, then that string specifies the *Roles* to which a user is assigned; this value is a string containing a comma-delimited list of roles. If no value is passed back to the calling routine, then there are no roles associated with the user account and the Management Portal indicates this. Information about a user's [roles](#) is available on the **Roles** tab of a user's **Edit User** page and a user's profile.

If any roles returned in *Properties("Roles")* are not defined, then the user is not assigned to the role.

Hence, the logged-in user is assigned to roles as follows:

- If a role is listed in *Properties("Roles")* and is defined by the InterSystems IRIS instance, then the user is assigned to the role.
- If a role is listed in *Properties("Roles")* and is not defined by the InterSystems IRIS instance, then the user is not assigned to the role.
- A user is always assigned to those roles associated with the `_PUBLIC` user. A user also has access to all public resources. For information on the `_PUBLIC` user, see [The _PUBLIC account](#); for information on public resources, see [Services and their resources](#).

Routine

If **ZAUTHORIZE** sets the value of *Properties("Routine")*, then that string becomes the value of the user account's *Startup Tag^Routine* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) If no value is passed back to the calling routine, then the value of *Startup Tag^Routine* for the user account is a null string and the relevant field in the Management Portal then holds no content.

If *Properties("Routine")* has a value, then this value specifies the routine to execute automatically following login on a terminal-type service (such as Console, Terminal, or Telnet). If *Properties("Routine")* has no value or a value of " ", then login starts the Terminal session in programmer mode, subject to whether they have the privilege to access programmer mode or not.

Username

If **ZAUTHORIZE** sets the value of *Properties("Username")*, then that string becomes the value of the user account's *Name* property in InterSystems IRIS. (This property is described in [User Account Properties](#).) This provides the application programmer with an opportunity to normalize content provided by the end-user at the login prompt (while ensuring that the normalized username only differ by case).

If there is no explicit call that passes the value of *Properties("Username")* back to the calling routine, then there is no normalization and the value entered by the end-user at the prompt serves as the value of the user account's *Name* property without any modification.

The User Information Repository

ZAUTHORIZE can refer to any kind of repository of user information, such as a global or an external file. It is up to the code in the routine to set any external properties in the *Properties* array so that the authenticated user can be created or updated with this information. For example, while a repository can include information such as roles and namespaces, **ZAUTHORIZE** code must make that information available to InterSystems IRIS.

If information in the repository changes, this information is only propagated back into the InterSystems IRIS user information if there is code in **ZAUTHORIZE** to perform this action. Also, if there is such code, changes to users' roles must occur in the repository; if you change a user's roles during a session, the change does not become effective until the next login, at which point the user's roles are reset by **ZAUTHORIZE**.

ZAUTHORIZE Return Value and Error Messages

The routine returns one of the following values:

- Success — **\$\$SYSTEM.Status.OK()**. This indicates that **ZAUTHORIZE** has successfully executed. Depending on the code in the routine, this can indicate successful authentication of the user associated with *Username* and *Password*, successful authorization of the user associated *Username*, or both.
- Failure — **\$\$SYSTEM.Status.Error(***ERRORMESSAGE)**. This indicates that authorization failed. When **ZAUTHORIZE** returns an error message, it appears in the audit log if the LoginFailure event auditing is enabled; the end-user only sees the **\$\$SYSTEM.Status.Error(***AccessDenied)** error message.

ZAUTHORIZE can return system-defined or application-specific error messages. All these messages use the **Error** method of the %SYSTEM.Status class. This method is invoked as **\$\$SYSTEM.Status.Error** and takes one or two arguments, depending on the error condition.

The available system-defined error messages are:

- **\$\$SYSTEM.Status.Error(***AccessDenied)** — Error message of "Access Denied"
- **\$\$SYSTEM.Status.Error(***InvalidUsernameOrPassword)** — Error message of "Invalid Username or Password"
- **\$\$SYSTEM.Status.Error(***UserNotAuthorizedOnSystem,Username)** — Error message of "User *username* is not authorized"
- **\$\$SYSTEM.Status.Error(***UserAccountIsDisabled,Username)** — Error message of "User *username* account is disabled"
- **\$\$SYSTEM.Status.Error(***UserInvalidUsernameOrPassword,Username)** — Error message of "User *username* invalid name or password"
- **\$\$SYSTEM.Status.Error(***UserLoginTimeout)** — Error message of "Login timeout"
- **\$\$SYSTEM.Status.Error(***UserCTRLC)** — Error message of "Login aborted"
- **\$\$SYSTEM.Status.Error(***UserDoesNotExist,Username)** — Error message of "User *username* does not exist"
- **\$\$SYSTEM.Status.Error(***UserInvalid,Username)** — Error message of "Username *username* is invalid"
- **\$\$SYSTEM.Status.Error(***PasswordChangeRequired)** — Error message of "Password change required"
- **\$\$SYSTEM.Status.Error(***UserAccountIsExpired,Username)** — Error message of "User *username* account has expired"
- **\$\$SYSTEM.Status.Error(***UserAccountIsInactive,Username)** — Error message of "User *username* account is inactive"

- **`$$SYSTEM.Status.Error($$UserInvalidPassword)`** — Error message of “Invalid password”
- **`$$SYSTEM.Status.Error($$$ServiceDisabled,ServiceName)`** — Error message of “Logins for Service *username* are disabled”
- **`$$SYSTEM.Status.Error($$$ServiceLoginsDisabled)`** — Error message of “Logins are disabled”
- **`$$SYSTEM.Status.Error($$$ServiceNotAuthorized,ServiceName)`** — Error message of “User not authorized for service”

To use these error codes, uncomment the `#include %occErrors` statement that appears in **`ZAUTHORIZE.mac`**.

To generate a custom message, use the **`$$SYSTEM.Status.Error()`** method, passing it the `$$$GeneralError` macro and specifying any custom text as the second argument. For example:

```
$$SYSTEM.Status.Error($$$GeneralError,"Any text here")
```

Note that when an error message is returned to the caller, it is logged in the audit database (if LoginFailure event auditing is turned on). However, the only error message the user sees is **`$$SYSTEM.Status.Error($$AccessDenied)`**. However, the user also sees the message for the `$$$PasswordChangeRequired` error. Return this error if you want the user to change from the current to a new password.

Configure an Instance to Use Delegated Authorization

Once you have created a customized **`ZAUTHORIZE`** routine, the next step is to enable it for the instance’s relevant services or applications. This procedure is:

1. Run the **`^SECURITY`** routine from the `%SYS` namespace in a Terminal or Console window.
2. In **`^SECURITY`**, choose **System parameter setup**; under that, choose **Edit authentication options**; and under that, choose either **Allow Kerberos authentication** or **Allow Operating System authentication**. (Delegated authorization is only supported for these two authentication mechanisms.).
3. If you have selected **Allow Operating System authentication**, choose **Allow Delegated Authorization for O/S authentication**. If you have selected **Allow Kerberos authentication**, choose **Allow Delegated Authorization for Kerberos authentication**.

Selecting either of these choices causes InterSystems IRIS to invoke the **`ZAUTHORIZE.mac`** routine, if one exists, in the `%SYS` namespace.

Important: InterSystems IRIS only calls **`ZAUTHORIZE`** after user authentication.

Delegated Authorization and User Types

When a user first logs in to InterSystems IRIS with an authentication mechanism that uses delegated authorization, the system creates a user account either of Type OS (for Operating System) or Kerberos. (Note that this value does not appear in the Type column of the table of users on the **Users** page (**System Administration > Security > Users**).) At the time of account creation and, for subsequent logins, the **`ZAUTHORIZE`** routine specifies the roles for the user.

Any attempt to log in without using delegated authorization will result in a login failure. This is because only delegated authorization specifies the user Type as OS or Kerberos. When using these authentication mechanisms without delegated authorization, the user is authenticated as being of the Password user type; the login fails because a user can only have one type and a user of one type cannot log in using mechanisms associated with another type. (Delegated authentication and LDAP authentication also both fail for the same reason.)

For general information about user types, see [About User Types](#).

After Authorization — The State of the System

If the user is successfully authorized, the InterSystems IRIS security database is updated in one of the following ways:

1. If this is the first time the user has logged in, a user record is created in the security database for the entered username, using properties returned by **ZAUTHORIZE**.
2. If the user has logged in before, the user record is updated in the security database, using properties returned by this function.

Whether for a first-time user or not, the process that logs in has the value of the *\$ROLES* system variable set to the value of `Properties("Roles")`. For a terminal login, the namespace is set to the value of `Properties("NameSpace")` and the startup routine is set to the value of `Properties("Routine")`.

Advanced Topics in Authentication

System Variables and Authentication

After authentication, two variables have values:

- `$USERNAME` contains the username
- `$ROLES` contains a comma-delimited list of the roles that the user holds

You can use the `$ROLES` variable to [manage roles programmatically](#).

Use Multiple Authentication Mechanisms

The one situation in which InterSystems recommends the use of multiple authentication mechanisms is when moving from a less rigorous mechanism to a more rigorous one. For example, if an instance has been using no authentication and plans to make a transition to Kerberos, the following scenario might occur:

1. For the transition period, configure all supported services to allow both unauthenticated and Kerberos-authenticated access. Users can then connect using either mechanism.
2. If appropriate, install new client software (which uses Kerberos for authentication).
3. Once the list of InterSystems IRIS users has been synchronized with that in the Kerberos database, shut off unauthenticated access for all services.

The use of multiple authentication mechanisms is often in conjunction with cascading authentication, described in the next section.

Cascading Authentication

While InterSystems IRIS supports for a number of different authentication mechanisms, InterSystems recommends that you do not use any other password-based authentication mechanism along with Kerberos. Also, there are limited sets of circumstances when it is advisable for an instance to have multiple authentication mechanisms in use.

If a [service](#) supports multiple authentication mechanisms, InterSystems IRIS uses what is called *cascading authentication* to manage user access. With cascading authentication, InterSystems IRIS attempts to authenticate users via the specified mechanisms in the following order:

- Kerberos cache (includes Kerberos with or without integrity-checking or encryption)
- OS-based
- LDAP (with checking the LDAP credentials cache second)
- Delegated
- Instance authentication
- Unauthenticated

Note: If a service specifies Kerberos prompting and this fails, there is no cascading authentication. If a service specifies both Kerberos prompting and Kerberos cache, then InterSystems IRIS uses Kerberos cache only.

For example, if a service supports authentication through:

1. Kerberos cache
2. OS-based
3. Unauthenticated

If a user attempts to connect to InterSystems IRIS, then there is a check if the user has a Kerberos ticket-granting ticket; if there is such a ticket, there is an attempt to obtain a service ticket for InterSystems IRIS. If this succeeds, the user gets in. If either there is no initial TGT or an InterSystems service cannot be obtained, authentication fails and, so, cascades downward.

If the user has an OS-based identity that is in the InterSystems IRIS list of users, then the user gets in. If the user's OS-based identity is not in the InterSystems IRIS list of users, then authentication fails and cascades downward again.

When the final option in cascading authentication is unauthenticated access, then all users who reach this level gain access to InterSystems IRIS.

Note: If an instance supports cascading authentication and a user is authenticated with the second or subsequent authentication mechanism, then there have been login failures with any mechanisms attempted prior to the successful one. If the %System/%Login/LoginFailure audit event is enabled, these login failures will appear in the instance's audit log.

Establish Connections with the UnknownUser Account

If instance authentication and unauthenticated mode are both enabled, then a user can simply press **Enter** at the **Username** and **Password** prompts to connect to the service in unauthenticated mode, using the UnknownUser account. If only instance authentication is enabled, then pressing **Enter** at the **Username** and **Password** prompts denies access to the service; InterSystems IRIS treats this as a user attempting to log in as the UnknownUser account and providing the wrong password.

Programmatic Logins

In some situations, it may be necessary for a user to log in after execution of an application has begun. For example, an application may offer some functionality for unauthenticated users and later request the user to log in before some protected functionality is provided.

An application can call the InterSystems IRIS log in functionality through the **Login** method of the \$SYSTEM.Security class with the following syntax:

ObjectScript

```
set success = $SYSTEM.Security.Login(username,password)
```

where

- *success* is a boolean where 1 indicates success and 0 indicates failure
- *username* is a string holding the name of the account logging in
- *password* is a string holding the password for the *username* account

If the username and password are valid and the user account is enabled and its expiration date has not been reached, then the user is logged in, \$USERNAME and \$ROLES are updated accordingly, and the function returns 1. Otherwise, \$USERNAME and \$ROLES are unchanged and the function returns 0.

No checking of privileges occurs as a result of executing \$SYSTEM.Security.Login. As a result, it is possible that the process has lost privileges that were previously held.

There is also a one-argument form of \$SYSTEM.Security.Login:

ObjectScript

```
set success = $SYSTEM.Security.Login(username)
```

It behaves exactly the same as the two-argument form except that no password checking is performed. The single-argument form of \$SYSTEM.Security.Login is useful when applications have performed their own authentication and want to set

the InterSystems IRIS user identity accordingly. It can also be used in situations where a process is executing on behalf of a specific user but is not started by that user.

Note: The single-argument form of the **Login** method is a [restricted system capability](#).

The JOB Command and Establishing a New User Identity

When a process is created using the JOB command, it inherits the security characteristics (that is, the values of *\$USERNAME* and *\$ROLES*) of the process that created it. Note that all roles held by the parent process, User as well as Added, are inherited.

In some cases, it is desirable for the newly created process to have *\$USERNAME* and *\$ROLES* values that are different from its parent's values. For example, a task manager might be created to start certain tasks at certain times on behalf of certain users. While the task manager itself would likely have significant privileges, the tasks should run with the privileges of the users on whose behalf they are executing, not with the task manager's privileges.

The following pseudocode illustrates how this can be done:

```
WHILE ConditionToTest {
  IF SomethingToStart {
    DO Start(Routine, User)
  }
}

Start(Routine, User) {
  NEW $ROLES      // Preserve $USERNAME and $ROLES

  // Try to change username and roles
  IF $SYSTEM.Security.Login(User) {
    JOB ...
    QUIT $TEST
  }
  QUIT 0          // Login call failed
}
```

Authentication and the Management Portal

The Management Portal consists of several separate [web applications](#). The main page of the Portal is associated with the */csp/sys* application and other pages are associated with various */csp/sys/** applications (such as the security-related content, which is associated with the */csp/sys/sec* application). If the applications do not all have a common set of authentication mechanism(s) in use, users going from one Portal page to another may encounter login prompts or sudden shifts in their level of privilege.

For example, if the */csp/sys* application is using instance authentication exclusively, while other related Portal applications are using unauthenticated access exclusively, then, as users move from one Portal page to another, they go from unauthenticated access to requiring authentication. Another possible case is this: the */csp/sys* application supports only instance authentication, the other applications support only unauthenticated access, and UnknownUser has no special privileges; in this case, when users go from the Portal's main page to its other pages, they may not have sufficient privileges to perform any action.

To check and configure the authentication mechanism for a web application, select the application from the **Web Applications** page in the Portal (**System Administration > Security > Applications > Web Applications**) and, for the displayed application, make selections under **Allowed Authentication Methods** as appropriate (typically, so that */csp/sys* and */csp/sys/** share a common set of authentication mechanisms).

Encryption

Introduction to InterSystems Encryption

Introduction to InterSystems Encryption

InterSystems IRIS® data platform includes support for managed key [encryption](#), a suite of technologies that protects data at rest. These technologies are:

- *Block-level database encryption*, also known simply as [database encryption](#) — A set of administrative tools to allow creation and management of databases in which all the data is encrypted. Such databases are managed through the Management Portal.
- *Data-element encryption for applications*, also known simply as [data-element encryption](#) — A programmatic interface that allows applications to include code for encrypting and decrypting individual data elements (such as particular class properties) as they are stored to and retrieved from disk.
- *Encryption key management*, also known simply as *key management* — A set of tools for creating and managing the keys that are used to encrypt either databases or data elements.

Keys for encrypting either databases or data elements are known as *data-encryption keys* and may also be known simply as *keys* (when the context is clear). Each instance can simultaneously have up to 256 data-encryption keys activated for database encryption and up to four data-encryption keys activated for data-element encryption; *activating* a key makes it available for encryption and decryption operations.

Note: You can simultaneously use a key in a key file for database encryption and data-element encryption.

InterSystems IRIS uses AES (the Advanced Encryption Standard) to perform its encryption and decryption when an instance writes to or reads from disk. For databases, InterSystems IRIS writes and reads in fixed-length blocks, and the entire database is encrypted, except for the single label block; this encrypted content includes the data itself, indexes, bitmaps, pointers, allocation maps, and incremental backup maps. For data elements, only the specified data is encrypted, and a unique identifier for the encryption key is included with the encrypted data on disk.

Encryption and decryption have been optimized, and their effects are both deterministic and small for any InterSystems IRIS platform. For information about how InterSystems IRIS database encryption affects facilities related to but separate from databases, see [Encryption and Database-Related Facilities](#).

Key Management Tasks

Key Management Tasks

A *key*, short for *data-encryption key*, is a 128-, 196-, or 256-bit bit string that is used with a cryptographic algorithm to reversibly encrypt or decrypt data. Each key has a unique identifier (known as a *GUID*), which InterSystems IRIS® data platform displays as part of key management activities.

Key management is the set of activities associated with creating keys, activating keys, deactivating keys, assigning default keys for various activities, and deleting keys. It also includes management activities associated with key storage. You can store keys in either of two ways:

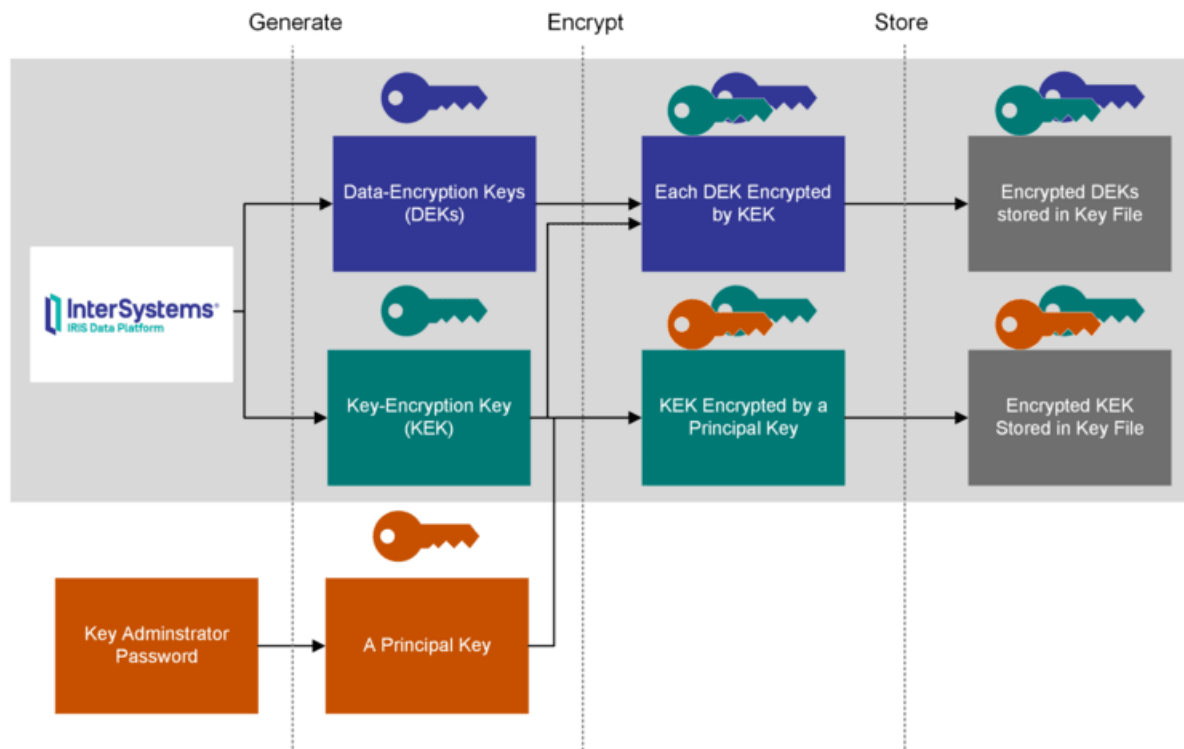
- In [key files](#) — The following types of key files are supported:
 - Standard Key Files — Key files that can contain multiple keys, the contents of which are encrypted via a key administrator passphrase. These key files can contain up to 256 encrypted keys. See [Manage Keys in Standard Key Files](#) for details.
 - KMS Key Files — Key files that contain keys encrypted via key management services (KMS) provided by the AWS or Azure cloud service provider (CSP). Each KMS key file can only contain a single key. See [Using a KMS for Key Management](#) for prerequisite and usage information.
- On a [KMIP server](#) — A *KMIP server* is a key management server that can send and receive communications using the key management interoperability protocol (KMIP). KMIP servers are available from various third-party vendors; those vendors provide instructions for configuring and using the KMIP server generally.

Note: If you wish to configure encryption for journal files or the IRISTEMP and IRISLOCALDATA databases, this is part of InterSystems IRIS startup configuration. See [Configure Encryption Startup Settings](#) for details.

Manage Keys in Standard Key Files

A *key file* is a file that holds encrypted copies of one or more *data-encryption keys* (DEKs). Key file management is the set of activities associated with key files themselves, such as adding administrators to or removing administrators from key files. Within a particular standard key file, all administrators have access to all keys. All keys are stored in an encrypted form, along with administrator information; each DEK is individually encrypted using a *key-encryption key* (KEK). For each administrator in the standard key file, there is a unique, encrypted copy of the KEK, which is encrypted using a *principal key* — where each principal key is derived from an individual key administrator's password. Encryption tasks require an activated DEK, and InterSystems IRIS requires an administrator username and password to decrypt that key so that it can then be used for encryption tasks. See the following diagram for a visual depiction of the standard key file process:

Standard Key File Encryption



Working with standard key files involves the following tasks:

- [Create a Standard Key File](#)
- [Add a Key to a Standard Key File, or Delete a Key from a Standard Key File](#)
- [Add an Administrator to a Standard Key File, or Delete an Administrator from a Standard Key File](#)
- [Activate a Database Encryption Key from a Standard Key File, or Deactivate a Database Encryption Key](#)
- [Activate a Data-Element Encryption Key from a Standard Key File, or Deactivate a Data-Element Encryption Key](#)
- [Manage Keys and Standard Key Files with Multiple-Instance Technologies](#)
- [Specify the Default Encryption Key or Journaling Encryption Key for an Instance](#)

Note: If an instance uses multiple keys at startup time (such as with journal files, the audit database, and other databases), then those keys must all be in a single standard key file. This allows them all to be available when the instance starts.

Create a Standard Key File

When you create a standard key file, it contains one key. To create a standard key file and its initial key:

1. From the Management Portal home page, go to the **Create Encryption Key File** page (**System Administration > Encryption > Create New Encryption Key File**).
2. On the **Create Encryption Key File** page, specify the following values:

- **Key File** — The name of the standard key file where the encryption key is stored; this can be an absolute or relative path name.

If you enter an absolute file name, the standard key file is placed in the specified directory on the specified drive; if you enter a relative file name, the standard key file is placed in the manager's directory for the InterSystems IRIS instance (which is below the InterSystems IRIS installation directory — that is, in <install-dir>/mgr/). Also, no file suffix is appended to the file name, so that the file MyKey is saved simply with that file name. You can also use the **Browse** button to the right of this field to choose the directory where InterSystems IRIS will create the standard key file. (If you provide the name of an existing file, InterSystems IRIS will not overwrite it and the save will fail.)

WARNING! Any key stored in <install-dir>/Mgr/Temp is deleted when InterSystems IRIS next reboots — *never* store a key in <install-dir>/Mgr/Temp.

- **Administrator Name** — The name of an administrator who can activate the key. There must be at least one administrator.

Because the database encryption functionality exists independent of InterSystems IRIS security, this name need not match any user names that are part of InterSystems IRIS security. By default, the initial administrator name value is the current username. The administrator name cannot include Unicode characters.

- **Password** — A password for this user.

Because the database encryption functionality exists independent of InterSystems security, this password need not match the password that a user has for InterSystems IRIS security. Note that this password is not stored anywhere on disk; it is the responsibility of the administrator to ensure that this information is not lost.

InterSystems suggests that this password follow the [administrator password strength](#) guidelines. If someone can successfully guess a valid password, the password policy is too weak. Also, this password cannot include Unicode characters.

Important: The key administrator's password is not stored anywhere on disk. It is the responsibility of the key administrator to ensure that this information is not lost.

- **Confirm Password** — The password for this user entered again to confirm its value.
- **Cipher Security Level** — The length of the key, where choices are 128-bit, 192-bit, and 256-bit.
- **Key Description** — Text that describes the key that is initially created and stored in the standard key file. This text appears in the **Description** column of the **Encryption Keys Defined in Key File** table.

3. Click **Save** at the top of the page to save the standard key file to disk.
4. Having just created a key, follow the instructions in [Protection from Accidental Loss of Access to Encrypted Data](#) to create and store a backup copy of the newly updated standard key file.

This creates a standard key file with a single database encryption key in it and with a single administrator. The page displays ID for the key, which is a string such as 9158980E-AE52-4E12-82FD-AA5A2909D029. The key ID is a unique identifier for the key which InterSystems IRIS displays here and on other pages. It provides a means for you to keep track of the key, regardless of its location. This is important because, once you save the standard key file, you can move it anywhere you choose; this means that InterSystems IRIS cannot track it by its location.

The key is encrypted using the key-encryption key (KEK), and there is a single copy of the KEK, which is encrypted using the administrator's principal key. You can add additional keys to the standard key file according to the instructions in [Add](#)

a [Key to a Standard Key File](#). You can add administrators to the standard key file according to the instructions in [Add an Administrator to a Standard Key File](#).

WARNING! InterSystems strongly recommends that you create and store a backup copy of the standard key file. Each time you create a database encryption key, it is a unique key that cannot be re-created. Using the same administrator and password for a new key still results in the creation of a different and unique key. If an unactivated key is lost and cannot be recovered, the encrypted database that it protected will be unreadable and its data will be *permanently lost*.

Add a Key to a Standard Key File

When using standard key files, there are two different ways to create a key:

- Create a standard key file. This causes InterSystems IRIS to create a key and place it in the file. To create a standard key file, see [Create a Standard Key File](#).
- Add a key to an existing standard key file, as described in this section.

To add a key to an existing standard key file:

1. From the Management Portal home page, go to the **Manage Encryption Key File** page (**System Administration** > **Encryption** > **Manage Encryption Key File**).
2. On the **Manage Encryption Key File** page, in the **Key File** field, enter the name of the standard key file to which you want to add and store the key; click **OK**. This displays information about that standard key file; at the bottom of the shaded area, the **Encryption Keys Defined in Key File** table displays a list of the one to 256 keys in the standard key file. If there are 255 or fewer keys in the file, you can create a new key and add it to the file.
3. Click the **Add** button below the **Encryption Keys Defined in Key File** table to add a key to the standard key file. This displays the **Add a New Encryption Key** screen.
4. In the **Add a New Encryption Key** screen, enter values in the following fields:
 - **Existing Administrator Name** — The name of an administrator associated with the standard key file. (Administrators associated with the file appear in the **Administrators Defined in Key File** table on the **Manage Encryption Key File** page.)
 - **Existing Administrator Password** — This administrator's password.
 - **Description** — Text to describe the key. This text appears in the **Description** column of the **Encryption Keys Defined in Key File** table.
5. Click **OK** to save the key to the standard key file. This displays information about it in the **Encryption Keys Defined in Key File** table, including its ID, which is a string such as 9158980E-AE52-4E12-82FD-AA5A2909D029. (The key ID is a unique identifier for the key which InterSystems IRIS displays here and on other pages. It provides a means for you to keep track of the key, regardless of its location. This is important because, once you save the standard key file, you can move it anywhere you choose; this means that InterSystems IRIS cannot track it by its location.)
6. Having just added a new key to the standard key file, follow the instructions in [Protection from Accidental Loss of Access to Encrypted Data](#) to create and store a backup copy of the newly updated standard key file.

WARNING! InterSystems strongly recommends that you create and store a backup copy of the standard key file. Each time you create a database encryption key, it is a unique key that cannot be re-created. Using the same administrator and password for a new key still results in the creation of a different and unique key. If an unactivated key is lost and cannot be recovered, the encrypted database that it protected will be unreadable and its data will be *permanently lost*.

Delete a Key from a Standard Key File

To delete a key from a standard key file:

1. From the Management Portal home page, go to the **Manage Encryption Key File** page (**System Administration > Encryption > Manage Encryption Key File**).
2. On the **Manage Encryption Key File** page, in the **Key File** field, enter the name of the standard key file from which you want to delete the key; click **OK**. This displays information about that standard key file; at the bottom of the shaded area, the **Encryption Keys Defined in Key File** table displays a list of the one to 256 keys in the standard key file. If there is one or more keys in the file, you can delete a key from the file.
3. In the table of keys, click **Delete** in the row for a key to delete that key. Clicking **Delete** displays a confirmation page for the action.

If the key's **Delete** button is not available, this is because the key is the default encryption key or the journal encryption key for the file. To delete the key, first specify that another key is the default encryption key or the journal encryption key for the file by clicking **Set Default** or **Set Journal** for the other key.

4. Click **OK** on the confirmation dialog to delete the key from the file.

WARNING! Before deleting the only existing copy of a key, it is critical that you are absolutely sure that there is no existing encrypted content that uses it. If there is no copy of the key that is required to decrypt data, the encrypted data that it protected will be unreadable and *permanently lost*.

Add an Administrator to a Standard Key File

To add an administrator to an existing standard key file:

1. From the Management Portal home page, go to the **Manage Encryption Key File** page (**System Administration > Encryption > Manage Encryption Key File**).
2. In the **Key File** field, enter the path and filename of the standard key file to open and click **OK**; you can also use the **Browse** button to look for the key. Once the Portal opens the standard key file, it displays a table with the administrators listed in the file; administrator names appear in all capital letters, regardless of how they were defined.
3. In the table of administrators, click **Add** to add a new administrator. This displays a page with the following fields:
 - **Existing Administrator Name** — The name of an administrator already in the file.
 - **Existing Administrator Password** — The password associated with the already existing administrator in the file.
 - **New Administrator Name** — The name of the new administrator to be added to the file. Because the encryption functionality is independent of InterSystems IRIS security, the administrator name need not match any user names that are part of InterSystems IRIS security. This user name cannot include Unicode characters.
 - **New Administrator Password** — The password for the new administrator. InterSystems suggests that this password follow the [administrator password strength](#) guidelines; also, this password cannot include Unicode characters. Because the encryption functionality is independent of InterSystems IRIS security, the password need not match the password that a user has for InterSystems IRIS security.
 - **Confirm New Administrator Password** — Confirmation of the password for the new administrator.

Complete these fields and click **OK**. You have now added a new administrator to the standard key file.

Once you have added the new administrator to the standard key file, you may wish to copy the standard key file, making sure that each copy is in a secure location. Further, InterSystems strongly recommends that you create multiple administrators for each key, one of which has the name and password written down and stored in a secure location, such as in a fireproof safe. However, if copies of the standard key file are made and later on, as an administrative function, a new administrator is added, only the copy of the standard key file with the new administrator will be up to date.

Note: When you add a new administrator to a standard key file, that administrator's password is permanently associated with the entry for the administrator name created in the file. Once assigned, passwords cannot be changed. If you wish to assign a new password, delete the entry in the standard key file for that administrator name and then create a new entry with the same name and a new password.

Delete an Administrator from a Standard Key File

To delete an administrator from a standard key file:

1. From the Management Portal home page, go to the **Manage Encryption Key File** page (**System Administration > Encryption > Manage Encryption Key File**).
2. In the **Key File** field, enter the path and filename of the key and click **OK**. This displays a table with the administrators listed in the file (as well as a table of encryption keys in the file).
3. In the table of administrators, click **Delete** next to an administrator to remove that administrator for the key. Clicking **Delete** displays a confirmation page for the action. (If there is only one administrator in the file, there is no **Delete** button, as it is not permitted to delete this administrator.)
4. Click **OK** to delete the administrator from the file.

Activate a Database Encryption Key from a Standard Key File

InterSystems IRIS supports up to 256 simultaneously activated keys for database encryption. To activate a key from a standard key file for database encryption:

1. From the Management Portal home page, go to the **Database Encryption** page (**System Administration > Encryption > Database Encryption**). If there are already any activated keys, the page displays a table listing these.
2. On this page, click **Activate Key**, which displays the fields for activating a key.
3. Enter values for the following fields:
 - **Key File** — The name of the file where the encryption key is stored. If you enter an absolute file name, InterSystems IRIS looks for the standard key file in the specified directory on the specified drive; if you enter a relative file name, InterSystems IRIS looks for the standard key file starting in the manager's directory for the InterSystems IRIS instance (which is below the InterSystems IRIS installation directory — that is, in <install-dir>/mgr/). You can also use the **Browse** button to display a dialog for opening the standard key file.
 - **Administrator Name** — The name of an administrator for this key, specified either when the key was [created](#) or [edited](#).
 - **Password** — The password specified for the named administrator.
4. Click the **Activate** button.

InterSystems IRIS then attempts to activate all the keys in the specified file. If there are not enough slots to activate all the keys in the file, then InterSystems IRIS opens as many keys as it can.

After key activation, the **Database Encryption** page displays the table of activated keys. For each key that InterSystems IRIS activates, the page adds the key to table of activated keys and displays the key's identifier. For each activated key, you can also perform various actions:

- **Set Default** — Click to specify that InterSystems IRIS uses this key when creating new encrypted databases. For more details, see [Specify the Default Encryption Key or Journaling Encryption Key for an Instance](#).
- **Set Journal** — Click to specify that InterSystems IRIS uses this key to encrypt journal files. For more details, see [Specify the Default Encryption Key or Journaling Encryption Key for an Instance](#).
- **Deactivate** — Click to deactivate this key. For more details, see [Deactivate a Database Encryption Key](#)

Note: The table of keys does not display any file or path information. This is because, once a standard key file is created, any sufficiently privileged operating system user can move it; hence, InterSystems IRIS may not have accurate information about the operating system location and can only rely on the accuracy of the GUID for the activated key in memory. When activating a second or subsequent key, note the identifier(s) for the currently activated key(s) first, so that you can identify the new one.

Activate a Data-Element Encryption Key from a Standard Key File

InterSystems IRIS supports up to four activated keys at one time for data-element encryption. To activate a key for data-element encryption:

1. From the Management Portal home page, go to the **Data Element Encryption** page (**System Administration** > **Encryption** > **Data Element Encryption**). If there are already any activated keys, the page displays a table listing these.
2. On the **Data Element Encryption** page, select **Activate Key**, which displays the fields for activating a key. If key activation is not available, this is because there are already four activated data element keys.
3. Enter values for the following fields:
 - **Key File** — The name of the file where the encryption key is stored. If you enter an absolute file name, InterSystems IRIS looks for the standard key file in the specified directory on the specified drive; if you enter a relative file name, InterSystems IRIS looks for the standard key file starting in the manager's directory for the InterSystems IRIS instance (which is below the InterSystems IRIS installation directory — that is, in <install-dir>/mgr/).
 - **Administrator Name** — The name of an administrator for this key, specified either when the key was [created](#) or [edited](#).
 - **Password** — The password specified for the named administrator.
4. Click the **Activate** button.

InterSystems IRIS then attempts to activate all the keys in the specified file. If there are not enough slots to activate all the keys in the file, then InterSystems IRIS opens as many keys as it can.

After key activation, the **Data Element Encryption** page displays the table of activated keys. For each key that InterSystems IRIS activates, the page adds the key to table of activated keys and displays the key's identifier.

Note: The table of keys does not display any file or path information. This is because, once the standard key file is activated, any sufficiently privileged operating system user can move the key; hence, InterSystems IRIS may not have accurate information about the operating system location and can only rely on the accuracy of the GUID for the activated key in memory. When activating a second or subsequent key, note the identifier(s) for the currently activated key(s) first, so that you can identify the new one.

Manage Keys and Standard Key Files with Multiple-Instance Technologies

If you are using encrypted databases or journal files within an InterSystems IRIS cluster, the InterSystems IRIS instances on all nodes in the cluster must share a single database encryption key.

Before enabling journal file encryption for any instance that belongs to an InterSystems IRIS mirror, see [Activating Journal Encryption in a Mirror](#) for important information. (There are no mirroring-related requirements in regard to database encryption.)

There are two ways to enable sharing of a single key:

- All of the instances share a single standard key file, which is located on one cluster node or mirror member.

In this case, if you change the single copy of the standard key file, then these changes are visible to all nodes or members. However, if the host holding the standard key file becomes unavailable to the other nodes or members, any attempt to read the key from the standard key file fails; this can prevent InterSystems IRIS instances from restarting properly.

- Each cluster node or mirror member has its own copy of the standard key file.

Here, if you change the standard key file, then you propagate copies of the standard key file (containing the same key) to all the other nodes or members. This increases the burden of administering the standard key file (which is typically small), but ensures that each instance of InterSystems IRIS always has a key available at startup.

Important: Whether there are single or multiple standard key files, the database encryption key itself is the same for all instances.

Using a Single Standard Key File

To use a single standard key file:

1. Create a database encryption key on one node or member. For more information on this procedure, see [Create a Standard Key File](#).
2. Secure this key according to the instructions in [Protection from Accidental Loss of Access to Encrypted Data](#).

CAUTION: Failure to take these precautions can result in a situation in which the encrypted databases or journal files are unreadable and *permanently lost*.

3. Configure each instance of InterSystems IRIS for unattended startup and provide InterSystems IRIS with the path to the standard key file. For more information on this procedure, see [Startup with Unattended Key Activation](#).

Since all the InterSystems IRIS instances use the same key, they are able to read data encrypted by each other. Any changes to the standard key file are visible to all instances.

Using Multiple Standard Key Files

To use multiple copies of a standard key file:

1. Create a database encryption key on one node or member. For more information on this procedure, see [Create a Standard Key File](#).
2. Secure this key according to the instructions in [Protection from Accidental Loss of Access to Encrypted Data](#).

CAUTION: Failure to take these precautions can result in a situation in which the encrypted databases or journal files are unreadable and *permanently lost*.

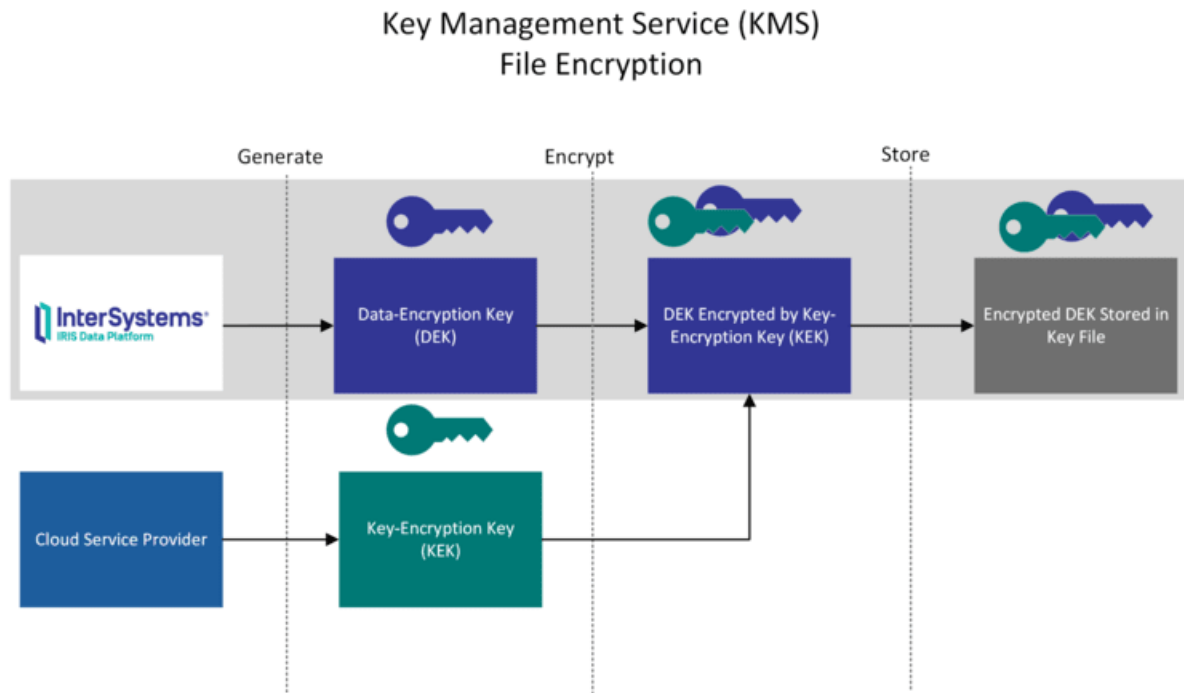
3. Make a copy of the standard key file for each of the remaining nodes or members.
4. On each node or member:
 - a. Get a copy of the standard key file and put it in a secure and stable location on that machine.
 - b. Configure each instance of InterSystems IRIS for unattended startup. For more information on this procedure, see [Startup with Unattended Key Activation](#).

Since each copy of the standard key file contains the same key, all the InterSystems IRIS instances are able to read data encrypted by each other. Since each InterSystems IRIS instance has a standard key file on its machine, the standard key file should always be available for an InterSystems IRIS restart. If there are any changes to the standard key file (such as adding or removing administrators), you must propagate new copies of the standard key file to each machine and reconfigure each instance of InterSystems IRIS for unattended startup using the new copy of the standard key file (even if that file is in the same location as the old file).

Using a KMS for Key Management

InterSystems IRIS enables you to use the key management services (KMS) provided by a cloud service provider (CSP). The KMS key acts as the key-encryption key (KEK), as in the standard key file encryption process. That is, the KEK

encrypts the data-encryption key (DEK) which is then stored in the KMS key file. A diagram showing an overview of this process is below:



Prerequisites

- Install the command line interface (CLI) specific to your KMS provider. See the KMS documentation provided by the CSP for specifics.
- Create and configure the user/service principal that represents the InterSystems IRIS instance when authenticating to the CSP. This user needs to have the appropriate key access policy configured through the key's associated key policy. For Azure, this is through the key's associated key vault access configuration, using either vault access policy or RBAC permission models. Check you can authenticate to the CSP through the CLI using the InterSystems IRIS user or service principal you configured. See the KMS documentation provided by the CSP for specifics.
- Create a KMS key on the CSP. For AWS, create a symmetric key. See [Creating Keys for AWS](#) for more details. For Azure, create an RSA key. See [About Azure Key Vault Keys](#) for more details. Keep the key ID of this key available as you need it for creating the KMS key file.

Interacting with the KMS

To perform key management tasks using the KMS for encryption, you can either invoke the `^EncryptionKey` routine or the InterSystems IRIS KMS API, both are in the `%SYS` namespace in the InterSystems IRIS terminal. For the `^EncryptionKey` routine, enter at the command line:

```
%SYS>do ^EncryptionKey
```

`^EncryptionKey` has a menu-driven interface that enables you to perform the following KMS key management tasks:

- Create a key.
- Activate a database encryption key.
- Activate a data element encryption key.
- Configure unattended key activation with a KMS key file.

See [Command-Line Security Management Utilities](#) for more information about ^EncryptionKey.

Creating a New KMS Encryption Key Using the ^EncryptionKey Routine

To create a new KMS encryption key using the ^EncryptionKey routine, follow these steps:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, switch to the %SYS namespace:

```
>zn "%SYS"
```
3. Run ^EncryptionKey:

```
>do ^EncryptionKey
```
4. In ^EncryptionKey, select option 1 **Create new encryption file**, then enter a name for your key file at the prompt. No extension is appended so the name you enter is the exact name of the file.
5. To use KMS, enter **y** at the **Use KMS?** prompt.
6. If desired, enter a description.
7. Select the key size.
8. Optional: If you decide to back up to a standard key file:
 - a. Enter the name of the backup standard key file and a key description when prompted.
 - b. Enter the username and password of a key administrator.

Because the database encryption functionality exists independent of InterSystems security, this password need not match the password that a user has for InterSystems IRIS security. Note that this password is not stored anywhere on disk; it is the responsibility of the administrator to ensure that this information is not lost.

InterSystems suggests that this password follow the [administrator password strength](#) guidelines. If someone can successfully guess a valid password, the password policy is too weak. Also, this password cannot include Unicode characters.

9. Choose the desired KMS (AWS or Azure).
10. At the **Server Key ID** prompt, enter the key ID of the KMS key you wish to use. This is the ID of the key you previously configured on the KMS.
11. If you are using Azure, skip this step. If you are using AWS:
 - a. Enter your region (for example, `us-east-2`).
 - b. Optional: You can enter AWS environment variables at the **Environment variable key** prompt. For each desired variable, enter the variable key at the **Environment variable key** prompt. Then enter the value at the **Environment variable value** prompt. See [the AWS documentation](#) for more information about AWS environment variables. One reason for using environment variables is to use a shared credentials file (`AWS_SHARED_CREDENTIALS_FILE`) for authentication to the CSP through the command line.

12. The new KMS key file and data-encryption key are created. The display will echo back the following information:

Encryption key file created

The name of the encryption file containing the data-encryption key.

Encryption key backup file created

The name of the backup standard key file (if provided).

Encryption key created via KMS

The key used to encrypt data in InterSystems IRIS.

13. Optional: Configure InterSystems IRIS startup options to use the new KMS key file by default:

- a. Choose 4 **Configure InterSystems IRIS startup options**.
- b. Choose 3 **Unattended key activation with a key file**, then enter the KMS key file name. If you are not using AWS, ignore the **Environment variable key** prompt.

For more information about unattended startup, see [Startup with Unattended Key Activation](#).

Creating a New KMS Encryption Key Using the InterSystems IRIS KMS API

To create a new KMS encryption key file using the API, use the `$System.Encryption.KMSCreateEncryptionKey()` method. It has the same KMS prerequisites as the `^EncryptionKey` routine. It follows the below usage:

```
%SYS>set KeyID =
$System.Encryption.KMSCreateEncryptionKey(File,Server,ServerKeyID,KeyLength,.Backup,Region,Description,.Env,.Status)
```

We retrieve and store the `KeyID` because it is required to identify and import the desired data-encryption key. This works for data-at-rest encryption and data-element encryption.

The method arguments are defined as follows:

- **File** — Required. Name of the key file to create.
- **Server** — Required. Name of the KMS CSP server. Currently accepted values are "AWS" and "Azure" (case insensitive).
- **ServerKeyID** — Required. Key ID of the principal key on the server.
- **KeyLength** — Required. Length in bytes of the data- and key-encryption keys. Must be 16, 24, or 32.
- **Backup** — Passed by reference, optional. Information for creating a backup key file. If specified, it must contain the following entries:
 - `Backup("File")` — Required. Name of the backup key file.
 - `Backup("Username")` — Required. Name of the initial encryption key administrator for the backup key file.
 - `Backup("Password")` — Required. Password for the initial encryption key administrator for the backup key file.
- **Note:** You should always obtain this value from a user prompt. You should never embed it in application code.
- `Backup("Desc")` — Optional. Description of the key.
- **Region** — Required for AWS only. Name of the region, for example `us-east-2`.
- **Description** — Optional. Description of the key.
- **Env** — Passed by reference, optional for AWS only. Environment variable information. For example, `Env("AWS_CONFIG_FILE")` or `Env("AWS_SHARED_CREDENTIALS_FILE")`.
- **Status** — Passed by reference, required. Returns the method status.

On success, this method returns the unique identifier of the new encryption key. See the below examples for AWS and Azure usage. Note, the API requires the calling user to have `%Admin_Secure:U` privileges as well as access to the `%SYS` namespace.

For AWS KMS API call:

```
%SYS>w ^KeyIDaws
604cae51-139d-4b88-b8ac-8b303446ebe7
%SYS>s
KeyID=$System.Encryption.KMCreateEncryptionKey("KMSTestAWS.key","AWS",^KeyIDaws,16,,,"us-east-2","aws
test key",,,.rc)
%SYS>w rc," ",KeyID
1 3C978FFD-DEA8-4393-A454-BC06B311D545
```

For Azure KMS API call:

```
%SYS>w ^KeyIDaz
https://test.vault.azure.net/keys/testkey/3ab1ba844c0b407c9b6063cefe5053dd
%SYS>s KeyID=$System.Encryption.KMCreateEncryptionKey("KMSTestAZ.key","azure",^KeyIDaz,16,,,,.rc)
%SYS>w rc," ",KeyID
1 8DC5555E-A464-4A59-9B3A-FD06857E5056
```

Managing Keys with the Key Management Interoperability Protocol (KMIP)

InterSystems supports the use of a KMIP server to manage database encryption keys. Using KMIP includes the following tasks:

- [Create, edit, or delete a KMIP Server Configuration](#)
- [List the KMIP Server Configurations](#)
- [List Details about a KMIP Server Configuration](#)
- [Create a Key on the KMIP Server](#) or [Delete a Key on the KMIP Server](#)
- [List the Keys on the KMIP Server](#)
- [Activate a Database Encryption Key from a KMIP Server](#) or [Deactivate a Database Encryption Key](#)
- [Activate a Data-Element Encryption Key from a KMIP Server](#) or [Deactivate a Data-Element Encryption Key](#)
- [Copy a Key from a KMIP Server to a Key File](#)
- [Specify the Default Encryption Key or Journaling Encryption Key for an Instance](#)

Note:

- InterSystems IRIS supports KMIP protocol versions 1.0–2.1.
- KMIP activities are not supported on macOS instances of InterSystems IRIS.

Create a KMIP Server Configuration

When establishing a connection between InterSystems IRIS and a KMIP server, you create a *KMIP server configuration*, which defines properties of the KMIP server and represents it within the InterSystems IRIS instance. To create a KMIP server configuration:

1. Set up the KMIP server according to its vendor's instructions.

CAUTION: When configuring a KMIP server, follow all proper backup procedures according to your vendor's instructions. If you do not have backup copies of your keys, you may lose data *permanently*.

Once you have set up the server, you can then set up the KMIP server configuration in InterSystems IRIS:

2. To set up a KMIP server configuration, you must have:
 - The certificate authority (CA) certificate for the KMIP server, which must be a trusted CA. You should receive this certificate from the vendor that provides the KMIP server or should obtain it according to instructions from that vendor.

- A public-key certificate and private key for each instance of InterSystems IRIS that will communicate with the KMIP server. The certificate must be issued by a trusted CA. You should receive this certificate and private key from the vendor that provides the KMIP server or should obtain them according to instructions from that vendor.
 - The following information about the KMIP server:
 - Its fully-qualified DNS name or IP address
 - The port number on which it accepts connections
 - The version of the KMIP protocol that it supports
 - Any TLS settings that it requires for its clients
3. On the InterSystems IRIS instance that will communicate with the KMIP server, create an TLS configuration that will represent the instance to the KMIP server:
 - a. In the Portal, go to the **SSL/TLS Configurations** page (**Home > System Administration > Security > SSL/TLS Configurations**).
 - b. On the **SSL/TLS Configurations** page, click the **Create New Configuration** button, which displays the **New SSL/TLS Configuration** page.
 - c. On the **New SSL/TLS Configuration** page, set up the TLS configuration. For the fields listed below, specify or select values as follows
 - **Enabled** — Select this check box.
 - **Type** — Select **Client**.

The values for other fields (**Server certificate verification**, the **This client's credentials** fields, and the **Cryptographic settings** fields) depend on the requirements of the KMIP server. The values for the **This client's credentials** fields depend on the client certificate, client private key, and CA certificate that you have received from the vendor that provides the KMIP server.

For more information on this creating an TLS configuration, see [Create or Edit a TLS Configuration](#).

4. Create the configuration to the KMIP server:
 - a. Start the Terminal and log in as a sufficiently privileged user.
 - b. At the terminal prompt, go to the %SYS namespace:


```
>set $namespace="%SYS"
```
 - c. Run ^SECURITY


```
%SYS>do ^SECURITY
```
 - d. In ^SECURITY, select option **14, KMIP server setup**.
 - e. In the **KMIP server setup** choices, select option **1, Create KMIP server**.
 - f. At the **Create KMIP server** prompts, specify values for the following:
 - **KMIP server to create?** — The name of the KMIP server configuration.
 - **Description?** — A text description.
 - **Server host DNS name?** — The fully-qualified DNS name or IP address of the KMIP server.
 - **TCP port number?** — The port number on which the KMIP server accepts connections.

- **OASIS KMIP protocol version?** — The number associated with your KMIP server's supported version of the protocol. This is part of the information that you have received from the vendor that provides the KMIP server.
- **SSL/TLS Configuration name?** — The name of the TLS configuration that you created in the previous step.

Note: This case of the value that you enter here must match that of the TLS configuration name as defined.

- **Non-blocking I/O?** — Whether or not connections to the KMIP server enable non-blocking I/O. InterSystems recommends **Yes**, which enables non-blocking I/O.

If non-blocking I/O is enabled, control returns to the application after the timeout specified at the **I/O timeout, in seconds?** prompt (below). If non-blocking I/O is disabled, control returns to the application after an operating-system timeout (which may not occur).

- **Auto-reconnect?** — Whether or not InterSystems IRIS reconnects with the KMIP server if the connection drops. InterSystems recommends that you select **No**; there is then no attempt to automatically reconnect if the connection drops.
- **I/O timeout, in seconds?** — The amount of time, in seconds, before a timeout occurs in the connection to the KMIP server. This is only relevant if the configuration has enabled non-blocking I/O.
- **Log KMIP messages?** — Whether or not InterSystems IRIS logs messages that it sends to the KMIP server. If messages are logged, InterSystems IRIS stores them in the <install-dir>/mgr/kmipcmd.log file.
- **Debug SSL/TLS?** — Whether or not InterSystems IRIS logs TLS debugging information. If information is logged, InterSystems IRIS stores it in the <install-dir>/mgr/kmipssl.log file.

- g. After the prompts for KMIP server properties, confirm that you wish to create the KMIP server at the **Confirm creation of KMIP server** prompt.

Note: InterSystems supports the use of multiple KMIP servers and the use of a single KMIP server that has multiple configurations. The most recently activated configuration is the default.

Edit a KMIP Server Configuration

To modify the values of the properties of an existing KMIP server configuration:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```
3. Run **^SECURITY**:

```
%SYS>do ^SECURITY
```
4. In **^SECURITY**, select option **14, KMIP server setup**.
5. In the **KMIP server setup** choices, select option **2, Edit KMIP server**.
6. At the **Edit KMIP server** prompt, enter the name of the configuration to edit.
7. **^SECURITY** then presents prompts for the same properties as when creating a KMIP server configuration; it uses the existing values for the configuration's properties as its defaults. Modify these values as required.
8. After the prompts for KMIP server properties, confirm any edits to the properties of the KMIP server at the **Confirm changes to KMIP server <servername>** prompt.

Delete a KMIP Server Configuration

To delete a KMIP server configuration:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```
3. Run **^SECURITY**

```
%SYS>do ^SECURITY
```
4. In **^SECURITY**, select option **14, KMIP server setup**.
5. In the **KMIP server setup** choices, select option **5, Delete KMIP server**.
6. At the **KMIP server to delete?** prompt, enter the name of the configuration to delete.
7. Confirm the deletion when prompted.

List the KMIP Server Configurations

To list an InterSystems IRIS instance's KMIP server configurations:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```
3. Run **^SECURITY**

```
%SYS>do ^SECURITY
```
4. In **^SECURITY**, select option **14, KMIP server setup**.
5. In the **KMIP server setup** choices, select option **3, List KMIP servers**.

^SECURITY then displays a list of any existing configurations to KMIP servers by name, whether or not they are currently in use.

List Details about a KMIP Server Configuration

To view details about a particular KMIP server configuration:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```
3. Run **^SECURITY**

```
%SYS>do ^SECURITY
```
4. In **^SECURITY**, select option **14, KMIP server setup**.
5. In the **KMIP server setup** choices, select option **4, Detailed list KMIP server**.
6. Enter the name of a KMIP server configuration at the *Display which KMIP configuration?* prompt.

^SECURITY then displays a list of the specified configuration's properties, along with each one's value.

Create a Key on the KMIP Server

To create a data-encryption key on a KMIP server:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```
3. Run **^EncryptionKey**:

```
%SYS>do ^EncryptionKey
```
4. In **^EncryptionKey**, select option **5, Manage KMIP server**.
5. When prompted, enter the name of the configuration for the KMIP server on which you wish to create a key.
6. At the next prompt, where you select the action you wish to take, select option **2, for Create new key on KMIP server**.
7. At the next prompt, select a key length.

The **^EncryptionKey** routine then creates the key and displays its key ID. Newly created keys are not activated by default; to activate the key, see [Activate a Database Encryption Key from a KMIP Server](#).

Important: InterSystems recommends that you record the key ID, so that you have this information available for future reference.

Delete a Key on the KMIP Server

To delete an encryption key on a KMIP server:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```
3. Run **^EncryptionKey**:

```
%SYS>do ^EncryptionKey
```
4. In **^EncryptionKey**, select option **5, Manage KMIP server**.
5. When prompted, enter the name of the configuration for the KMIP server on which you wish to delete the key.
6. At the next prompt, where you select the action you wish to take, select option **3, for Destroy existing key on KMIP server**.
7. The routine then lists the keys on the KMIP server and prompts for the key to delete. Specify a key at the **Select key** prompt.

WARNING! Before deleting the only existing copy of a key, it is critical that you are absolutely sure that there is no existing encrypted content that uses it. If there is no copy the key required to decrypt data, the encrypted data that it protected will be unreadable and will be *permanently lost*.

8. When prompted, confirm that you wish to delete the key.

The routine then deletes the key from the KMIP server.

List the Keys on the KMIP Server

To list the encryption keys on a KMIP server:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```
3. Run **^EncryptionKey**:

```
%SYS>do ^EncryptionKey
```
4. In **^EncryptionKey**, select option **5, Manage KMIP server**.
5. When prompted, enter the name of the configuration of the KMIP server for which you wish to list the key(s).
6. At the next prompt, select option **1**, for **List keys on KMIP server**.

The routine then displays a list of all the keys on the KMIP server.

Activate a Database Encryption Key from a KMIP Server

To activate a database encryption key from a KMIP server:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```
3. Run **^EncryptionKey**:

```
%SYS>do ^EncryptionKey
```
4. In **^EncryptionKey**, select option **3, Database encryption**.
5. In the **Database encryption** choices, select option **1, Activate database encryption keys**.
6. In the **Activate database encryption keys** choices, select option **2, Use KMIP server**.

Note: If this prompt does not appear, it is because the instance does not have any KMIP server configurations; see [Create a KMIP Server Configuration](#) for instructions on this process.

7. When prompted, enter the name of the configuration of the KMIP server from which you wish to activate the key.
8. The routine then lists the keys on the KMIP server and prompts for which key to activate. Specify a key at the **Select key** prompt.

The routine then activates the key, displaying its ID.

For each key that InterSystems IRIS activates, the **Database Encryption** page (**System Administration > Encryption > Database Encryption**) adds the key to table of activated keys and displays the key's identifier.

Note: The table of keys does not display any file or path information. When activating a second or subsequent key, note the identifier(s) for the currently activated key(s) first, so that you can identify the new one.

Activate a Data-Element Encryption Key from a KMIP Server

InterSystems IRIS supports up to four activated keys at one time for data-element encryption. To activate a key for data-element encryption from a KMIP server:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```

3. Run **^EncryptionKey**:

```
%SYS>do ^EncryptionKey
```

4. In **^EncryptionKey**, select option **4, Data element encryption for applications**.
5. In the **Data element encryption for applications** choices, select option **1, Activate data element encryption key**.
6. In the **Activate data element encryption key** choices, select option **2, Use KMIP server**.

Note: If this prompt does not appear, it is because the instance does not have any KMIP server configurations; see [Create a KMIP Server Configuration](#) for instructions on this process.

7. At the KMIP server prompt, enter the name of the configuration of the KMIP server from which you wish to activate the key.
8. The routine then lists the keys on the KMIP server and prompts for which key to activate. Specify a key at the **Select key** prompt.

The routine then activates the key, displaying its ID.

For each key that InterSystems IRIS activates, the **Data Element Encryption** page (**System Administration > Encryption > Data Element Encryption**) adds the key to table of activated keys and displays the key's identifier.

Note: The table of keys does not display any file or path information. When activating a second or subsequent key, note the identifier(s) for the currently activated key(s) first, so that you can identify the new one.

Copy a Key from a KMIP Server to a Key File

You can copy a database encryption key from a KMIP server to a key file. This allows you to make keys available both for backup and for recovery from a network or KMIP service outage. You can:

- [Create a database encryption key file with a copy of a key from a KMIP server](#)
- [Add a copy of a database encryption key from a KMIP server to an existing encryption key file](#)

Important: Always store encryption key files on removable devices that are kept in securely locked storage.

Create a Key File with a Copy of a Key from a KMIP Server

To create a key file and copy a key from a KMIP server to it:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```

3. Run **^EncryptionKey**:

```
%SYS>do ^EncryptionKey
```

4. In **^EncryptionKey**, select option **1, Create new encryption key file**.
5. At the prompts that follow, specify:
- The name of the key file (which is relative the <install-dir>/mgr/ directory)
 - A description of the key file
 - The name of an administrator for the key — this is a new administrator and can have a new name
 - The password (then confirmed) for that administrator — this is a new password, which can have any valid value

- Available cipher security levels — the length of the key used to encrypt keys stored in the file
6. At the next prompt, select option **2, Copy key from KMIP server**. **^EncryptionKey** then prompts for the key to copy to the file.
 7. At the **Select key** prompt, specify the number of the key to copy.

^EncryptionKey then creates the file with the administrator username and password that you specified, and places the selected key in that file.

Adding a Copy of a Key from a KMIP Server to an Existing Key File

To add a key from a KMIP server to an existing key file:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:


```
>set $namespace="%SYS"
```
3. Run **^EncryptionKey**:


```
%SYS>do ^EncryptionKey
```
4. In **^EncryptionKey**, select option **2, Manage existing encryption key file**.
5. At the **Encryption key file** prompt, enter the path and name of the key file to which you are adding a key. The path is relative to the <install-dir>/mgr/ directory.
6. At the next prompt, select option **5, Add encryption key**. At the prompts that follow:
 - a. Under **Existing administrator**, enter the **Username** and **Password** of an administrator for the key file.
 - b. Enter a description of the key that you are adding to the key file.
7. At the next prompt, select option **2, Copy key from KMIP server**. At the prompts that follow:
 - a. At the **KMIP server** prompt, enter the name of the KMIP server from which you are copying the key.
 - b. At the **Select key** prompt, specify the number of the key to copy.

^EncryptionKey then adds the selected key to selected key file.

Storage-Independent Key Management Tasks

Some tasks are the same for keys in files and keys on a KMIP server:

- [Deactivate a Database Encryption Key](#)
- [Deactivate a Data-Element Encryption Key](#)
- [Specify the Default Database Encryption Key or Journaling Encryption Key for an Instance](#)

Deactivate a Database Encryption Key

To deactivate a database encryption key:

1. From the Management Portal home page, go to the **Database Encryption** page (**System Administration > Encryption > Database Encryption**). If a key is currently activated, its identifier appears in the table of keys.
2. You cannot deactivate a key if it is the default key either for new encrypted databases or for encrypting journal files. If you wish to deactivate a key that is InterSystems IRIS is using for either of these activities, then you must select a

different key to be used for them. Do this by clicking **Set Default** or **Set Journal** for another key. Once the key is not in use for either of these activities, its **Deactivate** button will be available.

3. To deactivate the key, click **Deactivate** in its row.

Note: If it is not possible to deactivate the key for some other reason, the Portal displays an error message. InterSystems IRIS does not allow you deactivate a key under the following circumstances:

- The IRISTEMP and IRISLOCALDATA databases are encrypted.
- There is a currently-mounted encrypted database (other than IRISTEMP and IRISLOCALDATA) that is encrypted with this key.
- The key is currently in use to encrypt journal files. (Note that if you change the journal file encryption key, until you switch the journal file, InterSystems IRIS continues to use the old key for encryption.)

See below for information about how to address the underlying condition.

4. Click **OK** on the confirmation dialog to deactivate the key.

To deactivate the key, each underlying condition requires a different action:

- For any encrypted database except IRISTEMP and IRISLOCALDATA, dismount the database on the **Databases** page (**System Operation** > **Databases**). You can then deactivate the key.
- For IRISTEMP and IRISLOCALDATA, specify that these databases are not to be encrypted and then restart InterSystems IRIS. To do this, select **Configure Startup Settings** on the **Database Encryption** page; either you can choose not to activate a database encryption key at startup (in which case InterSystems IRIS turns off encryption for IRISTEMP and IRISLOCALDATA) or you can choose interactive or unattended database encryption key activation at startup (in which cases the choice whether or not to encrypt IRISTEMP and IRISLOCALDATA becomes available — choose **No**).
- For encrypted journal files, ensure that no encrypted journal file is required for recovery. This is described in [Encrypted Journal Files](#).

Deactivate a Data-Element Encryption Key

To deactivate a data-element encryption key:

1. From the Management Portal home page, go to the **Data Element Encryption** page (**System Administration** > **Encryption** > **Data Element Encryption**) page. If there are any activated keys, the page displays a table listing them.
2. In the table of activated keys, for the key you wish to deactivate, click **Deactivate**. This displays a confirmation dialog.
3. In the confirmation dialog, click **OK**.

When the **Data Element Encryption** page appears again, the row in the table for the deactivated key should no longer be present.

Specify the Default Database Encryption Key or Journal Encryption Key for an Instance

Each instance has a default database encryption key and a default journal encryption key. The instance sets the initial value for each of these when an administrator first activates a database encryption key; the key that is initially the default depends on the key(s) that are in the activated key file. These values are preserved across InterSystems IRIS shutdowns and restarts.

To specify a new key for either of these purposes:

1. From the Management Portal home page, go to the **Database Encryption** page (**System Administration** > **Encryption** > **Database Encryption**). This displays a table of currently activated encryption keys for the instance.
2. In the table of encryption keys:

- To specify a new default encryption key, click **Set Default** for that key. The **Set Default** button for the current default key is unavailable.
- To specify a new journal encryption key, click **Set Journal** for that key. The **Set Journal** button for the current journal encryption key is unavailable.

3. When prompted to confirm your action, click **OK**.

InterSystems IRIS then sets the selected key as the default or journal encryption key. If a key is either the default or journal encryption key, then it cannot be deleted (since it is required for operations on the InterSystems IRIS instance). Hence, specifying either of these for a key makes the key's **Delete** button unavailable.

Using Encrypted Databases

Using Encrypted Databases

To protect entire databases that contain sensitive information, InterSystems IRIS® data platform supports block-level database encryption (or, for short, database encryption). Database encryption is technology that allows you to create and manage databases that, as entire entities, are encrypted; it employs the InterSystems IRIS key management tools to support its activities.

When you create a database, you can choose to have it be encrypted; this option is available if there is a currently activated [key](#). Once you have created an encrypted database, you can use it in the same way as you would use an unencrypted database. The encryption technology is transparent and has a small and deterministic performance effect.

This topic describes how to create and manage encrypted databases. The database encryption functionality also supports the ability to encrypt the audit log and journal files. Both these features require access to the database encryption key at startup time, as described in [Configure Encryption Startup Settings](#).

Create an Encrypted Database

When creating a new database, you can specify that it is encrypted. However, before you can create an encrypted database, InterSystems IRIS must have an activated database encryption key. Hence, the procedure is:

1. [Activate a database encryption key](#).
2. From the Management Portal home page, go to the **Local Databases** page (**System Administration** > **Configuration** > **System Configuration** > **Local Databases**).
3. On the **Local Databases** page, select **Create New Database**. This displays the **Create Database** wizard.
4. On the second page of the wizard, in the **Encrypt Database?** box, select **Yes**. This causes InterSystems IRIS to create an encrypted database. On all the other pages of the wizard, choose database characteristics as you would when creating any database. (For more information on creating databases, see [Create Local Databases](#).)

Note: InterSystems IRIS also provides [encryption management](#) tools to encrypt unencrypted databases or decrypt encrypted databases, if this is necessary.

Establish Access to an Encrypted Database

To perform various operations, such as adding a database to a mirror, the database must be mounted. However, for an encrypted database to be mounted, its key must be activated. Hence, access to the database requires activating the key and mounting the database, and the procedure for this is:

1. [Activate the key](#).
2. From the Management Portal home page, go to the **Databases** page (**System Operation** > **Databases**).
3. On this page, for the database that you wish to mount, select the **Mount** button in the far right column of its row in the table of databases. After selecting **OK** on the confirmation screen, the database is mounted. If the key is not activated, InterSystems IRIS cannot mount the database and displays an error message.

You can now access the data within the database.

Close the Connection to an Encrypted Database

To close the connection to an encrypted database, the procedure is:

1. From the Management Portal home page, go to the **Databases** page (**System Operation** > **Databases**).

2. On this page, select the **Dismount** button on the right in the table of databases. After selecting **OK** on the confirmation screen, the database is dismounted.
3. [Deactivate the key](#).

Because the activated key is used for each read and write to the database, you cannot deactivate the key without first dismounting the database. If you attempt to deactivate the key prior to dismounting the database, InterSystems IRIS displays an error message.

Move an Encrypted Database Between Instances

If your organization has multiple InterSystems IRIS instances, you may need to use an encrypted database on one instance that was created on another instance using a different key. To move the data from one instance to another, back up and then re-encrypt the database using the available encryption management tools. For more information, see [Modify Database Encryption Using ^EncryptionKey](#).

Configure Encryption Startup Settings

This topic describes how to set up each of the three database encryption startup options:

- [Startup without Key Activation](#) (the default) — The instance does not have a database encryption key available at startup time.
- [Startup with Interactive Key Activation](#) — The instance gathers database encryption key information at startup time interactively.
- [Startup with Unattended Key Activation](#) — The instance gathers database encryption key information at startup time without human intervention. This is also known as *unattended startup*.

InterSystems IRIS has several features that require having a key available at startup time (either interactively or through unattended startup):

- Encrypting the InterSystems IRIS audit log.
- Encrypting the IRISTEMP and IRISLOCALDATA databases. (Either both are encrypted or neither.)
- Encrypting InterSystems IRIS journal files.
- Having an encrypted database mounted at startup time.

Startup without Key Activation

This is the default behavior for an instance of InterSystems IRIS prior to having any keys activated. If you have set up key activation at startup and choose to turn it off, the procedure is:

1. From the Management Portal home page, go to the **Database Encryption** page (**System Administration > Encryption > Database Encryption**).
2. Select **Configure Startup Settings**. This displays the area with options for configuring InterSystems IRIS startup and other options for encrypted databases.
3. In this area, from the **Startup Options** list, select **None**.
4. Click **Save**. InterSystems IRIS may prevent you from performing this action if:
 - Any encrypted databases are required at startup. See [Encrypted Databases Required at Startup](#) for more details.
 - There are any encrypted journal files with open transactions. See [Encrypted Journal Files](#) for more details.
 - The audit log is encrypted. (The error message for this refers to an encrypted database because InterSystems IRIS stores the audit log in a database called IRISAUDIT.) See [Encrypted Audit Log](#) for more details.

Address the issue that is preventing the change and then perform this procedure again. Once any issues are corrected, you will be able to successfully change to having startup without key activation.

Encrypted Databases Required at Startup

If the instance has encrypted databases that are required at startup and you attempt to configure startup not to involve key activation, the Management Portal displays an error message stating this and indicating that the key activation option cannot be changed. (If the error message refers to the IRISAUDIT database, this is because the [audit log](#) is encrypted.)

To configure InterSystems IRIS to start without activating an encryption key, any encrypted databases can only be mounted after startup. To configure a database to be mounted after startup:

1. Confirm that the database is mounted or mount it:
 - a. From the Management Portal home page, go to the **Databases** page (**System Operation** > **Databases**).
 - b. Find the database's row in the table of databases. If it is mounted, there is a **Dismount** choice in its row; if it is not mounted, there is no **Dismount** choice and there *is* a **Mount** choice.
 - c. If it is not mounted, select **Mount**
 - d. On the confirmation screen, select **OK**. (The database needs to be writable, so do not select the **Read Only** check box.)
2. Edit the database's properties so that it is not mounted at startup:
 - a. Go to the **Local Databases** page (**System Administration** > **Configuration** > **System Configuration** > **Local Databases**).
 - b. Find the database's row in the table of databases.
 - c. Select the database by clicking on its name. This displays the page for editing the database.
 - d. On this **Edit** page, clear the **Mount Required at Startup** check box.
 - e. Click **Save**.

The database will no longer be mounted at startup. This means that it will no longer require key activation at startup (though it may be required for other reasons.)

Encrypted Journal Files

If the instance uses journaling and you attempt to configure startup not to involve key activation, you may be unable to turn off key activation at startup. These conditions are:

- The instance is configured to encrypt its journal files.
- There are open transactions in the journal file (which is fairly likely on a busy system).

If this occurs, you need to suspend the use of encrypted journal files before you change the startup key activation settings. To do this, the procedure is:

1. On the **Database Encryption** page (**System Administration** > **Encryption** > **Database Encryption**), change the **Encrypt Journal Files** setting to **No**. Leave the **Key Activation at Startup** setting as it is.
2. Switch journal files. To do this, click **Switch Journal** on the **Journals** page (**System Operation** > **Journals**).

Once all open transactions within the encrypted journal files have either been committed or rolled back, you can then change the InterSystems IRIS startup configuration.

CAUTION: Even after there are no open transactions, you may need the encrypted journal files to restore a database. For this reason, it is very important that you maintain copies of the key file containing the key used to encrypt these files.

For more information on journal files generally, see [Journaling](#).

Encrypted Audit Log

If the instance has an encrypted audit log and you attempt to configure startup not to involve key activation, InterSystems IRIS displays an error message that an encrypted database is required at startup, such as:

```
ERROR #1217: Can not disable database encryption key activation at startup.
Encrypted databases are required at startup:
C:\InterSystems\IRIS\Mgr\IRISAudit\
```

The error message refers to encrypted databases because the audit log is stored in the InterSystems IRIS database IRISAUDIT.

The audit log cannot be encrypted if InterSystems IRIS starts without activating an encryption key. To configure startup not to involve key activation, you must change the InterSystems IRIS setting to specify that the instance uses an unencrypted audit log. The procedure is:

1. Back up the instance's audit data.
2. Go to the **Database Encryption** page (**System Administration** > **Encryption** > **Database Encryption**).
3. Select **Configure Startup Settings**, which displays the area with options for configuring InterSystems IRIS startup and other options for encrypted databases.
4. Under **Optionally Encrypted Data**, in the **Encrypt Audit Log** list, click **No**.

Changing this setting causes InterSystems IRIS to erase any existing audit data, to start using unencrypted auditing immediately, and to write an AuditChange event to the audit log.

CAUTION: If you have not backed up audit data, changing the encryption setting for the audit log results in the loss of that existing audit data.

Startup with Interactive Key Activation

This is the default behavior for an instance of InterSystems IRIS if a key has been activated. With interactive key activation, the InterSystems IRIS instance prompts for the location of a key and its associated information during its startup.

Important: On Windows, interactive key activation is incompatible with configuring InterSystems IRIS as a service that starts automatically as part of system startup.

To configure InterSystems IRIS for interactive key activation:

1. From the Management Portal home page, go to the **Database Encryption** page (**System Administration** > **Encryption** > **Database Encryption**).
2. Select **Configure Startup Settings**. This displays the **Startup Options** area, which includes the **Key Activation at Startup** list.
3. In the **Key Activation at Startup** list, select **Interactive**. If the previous value for the field was **None**, then this displays the page's **Optionally Encrypted Data** area.
4. The fields in this area are:
 - **Encrypt IRISTEMP and IRISLOCALDATA Databases** — Allows you to specify whether or not the IRISTEMP and IRISLOCALDATA databases are encrypted. To encrypt them, select **Yes**; to have them be unencrypted, select **No**.
 - **Encrypt Journal Files** — Allows you to specify whether or not the instance encrypts its own journal files. To encrypt journal files, select **Yes**; to have them be unencrypted, select **No**. This choice depends on startup options because InterSystems IRIS startup creates a new journal file; if you choose encryption, startup requires a key.

Note: This change takes effect the next time that InterSystems IRIS switches journal files. To begin journal file encryption without a restart, switch journal files after completing this page.

- **Encrypt Audit Log** — Allows you to specify whether or not InterSystems IRIS encrypts the audit log. To encrypt the audit log, select **Yes**; to have it be unencrypted, select **No**. This choice depends on startup options because the InterSystems IRIS startup procedure records various events in the audit log; if you choose encryption, startup requires a key.

CAUTION: This change takes effect immediately and deletes any existing audit data. Back up the audit database prior to changing this setting; otherwise, audit data will be lost.

5. Click **Save** to save the selected settings.

Important: If InterSystems IRIS is configured to

- Encrypt IRISTEMP and IRISLOCALDATA, journal files, or the audit log
- Require an encrypted database at startup

then failure to activate the required encryption key causes an InterSystems IRIS startup failure. If this occurs, use InterSystems IRIS [emergency startup mode](#) to configure InterSystems IRIS not to require any encrypted facilities at startup.

Startup with Unattended Key Activation

Startup with unattended key activation, also known as *unattended startup*, activates a key and potentially mounts encrypted databases at startup time without any human intervention. Successful unattended startup requires that the instance have access to:

- The encrypted database
- The database encryption key, either through:
 - The KMIP server that holds the key
 - The database encryption key file that holds the key and the username and password used for unattended database encryption key activation

This section includes the following topics:

- [Configuring Unattended Startup Using a Key on a KMIP Server](#)
- [Configuring Unattended Startup Using a Key in a Key File](#)
- [Temporarily Addressing Issues with Unattended Startup](#)

CAUTION: By making all these items available, the security of the data in InterSystems IRIS becomes entirely dependent on the physical security of the machine(s) holding these elements. If your site cannot ensure this physical security, your data will then be subject to the same level of risk as if it were not encrypted; to avoid this situation, either use interactive startup (which prevents the simultaneous exposure of these elements) or ensure the physical security of the relevant machine(s).

Configuring Unattended Startup Using a Key on a KMIP Server

To configure an InterSystems IRIS instance for unattended startup using a key on a KMIP server:

1. For the relevant instance, start the Terminal and log in as a sufficiently privileged user.
2. At the terminal prompt, go to the %SYS namespace:

```
>set $namespace="%SYS"
```

3. Run **^EncryptionKey**:

```
%SYS>do ^EncryptionKey
```

4. In **^EncryptionKey**, select option **3, Database encryption**.5. At the next prompt, select option **4, Configure startup options**.6. At the next prompt, select option **4, Unattended key activation with a KMIP server**.7. At the **KMIP server instance name** prompt, enter the name of a KMIP server configuration.

8. At the prompts that follow, specify what items to encrypt (all of which require an activated key at startup time):

- **Encrypt journal files** (no by default) — Allows you to specify whether or not the instance encrypts its own journal files. To encrypt journal files, enter **yes**; to have them be unencrypted, enter or select **no** (the default). This choice depends on startup options because InterSystems IRIS startup creates a new journal file; if you choose encryption, startup requires a key.

This change takes effect the next time that InterSystems IRIS switches journal files. By default, this occurs the next time that InterSystems IRIS restarts. To begin journal file encryption without a restart, switch journal files after completing this page.

- **Encrypt IRISTEMP and IRISLOCALDATA databases** (no by default) — Allows you to specify whether or not the IRISTEMP and IRISLOCALDATA databases are encrypted. To encrypt them, enter **yes**; to have them be unencrypted, enter or select **no** (the default).
- **Encrypt audit database** (no by default) — Allows you to specify whether or not InterSystems IRIS encrypts the audit log. To encrypt the audit log, select **yes**; to have it be unencrypted, select **no** (the default). This choice depends on startup options because the InterSystems IRIS startup procedure records various events in the audit log; if you choose encryption, startup requires a key.

CAUTION: This change takes effect immediately and deletes any existing audit data. Back up the audit database prior to changing this setting; otherwise, audit data will be lost.

9. The routine then displays the current list of KMIP keys to activate at startup, and then prompts for the next action:

- To add a key to the list of the startup keys, select option **1, Add key to list**.
- To remove a key from the list of the startup keys, select option **2, Delete key from list**.
- To save the list of startup keys, select option **3, Save list**.

10. When the list contains the desired list of KMIP keys to activate at startup, select option **3**, which saves the list.

Configuring Unattended Startup Using a Key in a Key File

CAUTION: When you configure InterSystems IRIS for unattended startup, the instance adds another administrator to the database encryption key file; that administrator has a system-generated name and password. Once InterSystems IRIS has modified the key file to add this username and password, InterSystems strongly recommends that you place any copies of the key file only on hardware that can be physically locked in place, such as a lockable CD-ROM or DVD drive in a rack. Further, you should lock and monitor the data center facility where this hardware is stored. Do *not* store the database encryption key on the same drive as any databases that it is used to encrypt.

To configure an InterSystems IRIS instance for unattended startup with a key in a key file:

1. You need to have a key currently activated. To activate a key, see [Activating a Key](#).

2. From the Management Portal home page, go to the **Database Encryption** page (**System Administration > Encryption > Database Encryption**).
3. Select **Configure Startup Settings**. This displays the **Startup Options** list.
4. In **Startup Options**, select **Unattended (NOT RECOMMENDED)**. This changes the fields that the page displays.
5. The **Startup Options** area expands to display three fields. Complete these:
 - **Key File** — The path of the database encryption key file. This can be an absolute or relative path; if you specify a relative path, it is relative to the InterSystems IRIS installation directory. Click **Browse** to search for the database encryption key file on the file system.
 - **Administrator Name** — An administrator for this key file.
 - **Password** — The administrator's password.
6. Complete the fields in the **Optionally Encrypted Data** area:
 - **Encrypt IRISTEMP and IRISLOCALDATA Databases** — Allows you to specify whether or not the IRISTEMP and IRISLOCALDATA databases are encrypted. To encrypt them, select **Yes**; to have them be unencrypted, select **No**.
 - **Encrypt Journal Files** — Allows you to specify whether or not the instance encrypts its own journal files. To encrypt journal files, select **Yes**; to have them be unencrypted, select **No**. This choice depends on startup options because InterSystems IRIS startup creates a new journal file; if you choose encryption, startup requires a key.

Note: This change takes effect the next time that InterSystems IRIS switches journal files. By default, this occurs the next time that InterSystems IRIS restarts. To begin journal file encryption without a restart, switch journal files after completing this page.
 - **Encrypt Audit Log** — Allows you to specify whether or not InterSystems IRIS encrypts the audit log. To encrypt the audit log, select **Yes**; to have it be unencrypted, select **No**. This choice depends on startup options because the InterSystems IRIS startup procedure records various events in the audit log; if you choose encryption, startup requires a key.

CAUTION: This change takes effect immediately and deletes any existing audit data. Back up the audit database prior to changing this setting; otherwise, audit data will be lost.
7. Click **Save** to save the selected settings.

Temporarily Addressing Issues with Unattended Startup

If InterSystems IRIS is configured to

- Encrypt IRISTEMP and IRISLOCALDATA, journal files, or the audit log
- Require an encrypted database at startup

then failure to activate the encryption key causes an InterSystems IRIS startup failure. If this occurs, use InterSystems IRIS [emergency startup mode](#) to configure InterSystems IRIS not to require any encrypted facilities at startup.

Encrypt the Databases that Ship with InterSystems IRIS

Each instance of InterSystems IRIS ships with a number of databases. The ability to encrypt and the value of encryption depends on the database:

- **IRISLOCALDATA:** Can be encrypted in conjunction with the IRISTEMP database. Encrypting IRISLOCALDATA requires that a key be available at startup, since the database is required at startup time.

- **IRISAUDIT:** Can be encrypted. Encrypting IRISAUDIT requires that a key be available at startup, since the database is required at startup time.
- **IRISLIB:** Must not be encrypted. (Note that all content in IRISLIB is publicly available.)
- **IRISSYS:** Must not be encrypted. If an instance has an encrypted form of this database, InterSystems IRIS cannot start.
- **IRISTEMP:** Can be encrypted in conjunction with the IRISLOCALDATA database. Encrypting IRISTEMP requires that a key be available at startup, since the database is required at startup time.
- **USER:** Can be encrypted.

Modify Database Encryption Using ^EncryptionKey

There are occasions when you may need to perform encryption management operations that are not available through the Management Portal. Using the **^EncryptionKey** utility, you can perform the following actions:

- [Convert an Unencrypted Database to Be Encrypted](#)
- [Convert an Encrypted Database to Be Unencrypted](#)
- [Convert an Encrypted Database to Use a New Key](#)

The following is true about the tools used by the **^EncryptionKey** utility:

The **^EncryptionKey** utility uses a set of encryption management tools:

- When built-in hardware instructions are available for encryption-related activities, these activities are considerably faster than when using software-based encryption. The encryption management tools use hardware instructions when they are available.
- The encryption management tools can use keys stored on a KMIP server.
- The encryption management tools can run in [FIPS mode](#).

Note: The encryption management tools do not operate on journal files.

Convert an Unencrypted Database to be Encrypted

To convert an unencrypted database to an encrypted database:

1. Back up the data in the database to be encrypted.

InterSystems IRIS encrypts data in place. This means that it uses on-disk space for its operations (not copying the database elsewhere and restoring it to its current disk location after successful completion). If the utility is interrupted before completion, the database will be partly encrypted and partly unencrypted, rendering it unusable.

CAUTION: It is critical that you back up the database before converting it. Failure to do so can result in data being lost.

2. Activate the key with which you wish to encrypt the database, either from a [key file](#) or a [KMIP server](#).
3. Start the Terminal.
4. In the %SYS namespace, run the **^EncryptionKey** utility.
5. In **^EncryptionKey**, select option **3, Database encryption**.
6. In the database encryption submenu, select option **7, Modify encrypted status of existing database**.
7. In the **Database directories** submenu, select the database that you wish to modify; databases are listed by their directories. When you select a database, the routine announces if the database is encrypted or not.

8. If the database is unencrypted, the routine allows you to encrypt it; at the **Encrypt database?** prompt, enter *yes* or *y*. This is not case sensitive.
9. At the **Select key for encryption prompt**, select the key that the routine will use to encrypt the database. If the database is currently mounted, the routine then displays this information.
10. If the database is currently mounted, the routine states this. At the **Dismount database** prompt, enter *yes* or *y*. This is not case sensitive.

Important: Because dismounting and then remounting a database interrupts its operations, take the appropriate precautions to ensure that this does not cause problems.

The routine then encrypts the database. As part of this process, if the database was mounted, the routine displays messages that it has dismounted and mounted the database. When the database is mounted again, encryption is complete.

Convert an Encrypted Database to be Unencrypted

To convert an encrypted database to an unencrypted database:

1. Back up the data in the database to be unencrypted.

InterSystems IRIS unencrypts data in place. This means that it uses on-disk space for its operations (not copying the database elsewhere and restoring it to its current disk location after successful completion). If the utility is interrupted before completion, the database will be partly encrypted and partly unencrypted, rendering it unusable.

CAUTION: It is critical that you back up the database before converting it. Failure to do so can result in data being lost.

2. Activate the key with which you wish to encrypt the database, either from a [key file](#) or a [KMIP server](#).
3. Start the Terminal.
4. In the %SYS namespace, run the **^EncryptionKey** utility.
5. In **^EncryptionKey**, select option **3, Database encryption**.
6. In the database encryption submenu, select option **7, Modify encrypted status of existing database**.
7. In the **Database directories** submenu, select the database that you wish to modify; databases are listed by their directories. When you select a database, the routine announces if the database is encrypted or not. If the database is encrypted and its encryption key has not been activated, the routine announces this as well.
8. If the database is encrypted, the routine allows you to decrypt it; at the **Decrypt database?** prompt, enter *yes* or *y*. This is not case sensitive.
9. After reporting the encryption key for the database, the routine prompts if you wish to encrypt the database with a different key. Press Enter to simply convert it to a decrypted database and use a new key to encrypt it.
10. If the database is currently mounted, the routine states this. At the **Dismount database** prompt, enter *yes* or *y*. This is not case sensitive.

Important: Because dismounting and then remounting a database interrupts its operations, take the appropriate precautions to ensure that this does not cause problems.

The routine then decrypts the database. As part of this process, if the database was mounted, the routine displays messages that it has dismounted and mounted the database. When the database is mounted again, decryption is complete.

Convert an Encrypted Database to Use a New Key

To convert an encrypted database to use a new key:

1. Back up the data in the database to be re-encrypted.

InterSystems IRIS encrypts data in place. This means that it uses on-disk space for its operations (not copying the database elsewhere and restoring it to its current disk location after successful completion). If the utility is interrupted before completion, the database will be partly encrypted and partly unencrypted, rendering it unusable.

CAUTION: It is critical that you back up the database before converting it. Failure to do so can result in data being lost.

2. Activate the keys with which the database is encrypted and with which you wish to re-encrypt the database, either from a [key file](#) or a [KMIP server](#).
3. Start the Terminal.
4. In the %SYS namespace, run the **^EncryptionKey** utility.
5. In **^EncryptionKey**, select option **3, Database encryption**.
6. In the database encryption submenu, select option **7, Modify encrypted status of existing database**.
7. In the **Database directories** submenu, select the database that you wish to modify; databases are listed by their directories. When you select a database, the routine announces if the database is encrypted or not.
8. If the database is encrypted, the routine allows you to decrypt it; at the **Decrypt database?** prompt, enter **yes** or **y**. This is not case sensitive.
9. At the next prompt, which is the **Re-encrypt database?** prompt, enter **yes** or **y**. This is not case sensitive.
10. At the **Select key for encryption prompt**, select the key that the routine will use to encrypt the database.
11. If the database is currently mounted, the routine states this. At the **Dismount database** prompt, enter **yes** or **y**. This is not case sensitive.

Important: Because dismounting and then remounting a database interrupts its operations, take the appropriate precautions to ensure that this does not cause problems.

The routine then re-encrypts the database. As part of this process, if the database was mounted, the routine displays messages that it has dismounted and mounted the database. When the database is mounted again, encryption is complete.

Change Encryption Keys

You can change the encryption key that InterSystems IRIS uses for encryption.

Change Journal Encryption Key

You can change the encryption key InterSystems IRIS uses for journal file encryption. To rotate journal encryption keys on an instance, you must have both the current encryption key and the new encryption key activated simultaneously while the journal file switches. The following steps details how to change the journal encryption key:

Assume that EK1 is the current encryption key used as the default encryption key for journaling and that EK2 is the new encryption key you want to switch to.

1. Activate EK2 (**System Administration > Encryption > Database Encryption > Activate Key**).
2. Set EK2 as the default key for journals (**Set Journal** on Database Encryption page).
3. Switch the active journal file so that the newly created journal file uses EK2.
4. Once you no longer need the journal files encrypted with EK1, you also no longer need EK1.

You can check which encryption key each encrypted journal file uses via **System Operation > Journals > Summary** for the relevant file.

Data-Element Encryption

Using Data-Element Encryption

Data-element encryption provides a means of encrypting application data at a finer level of granularity than the database as a whole; it is for sensitive data elements whose exposure must be prevented. For example, customer records can exclusively encrypt the credit card field; patient records can exclusively encrypt fields that display test results (such as for HIV testing); or a record that includes a social security number can exclusively encrypt that field.

Data-element encryption is available programmatically (via an API), rather than through the Management Portal. Because it is accessible through an API, you can use it in your application code. You have the option of using data-element encryption with database encryption (though there is no requirement to use both).

For an application to use data-element encryption, the required keys must be available when the application is running. To make a key available, activate it; for details, either see [Programmatically Manage Keys](#) or, if using the Portal, see [Activating a Key for Data-Element Encryption](#). When a key is activated, InterSystems IRIS® data platform displays its unique identifier in the table of activated keys; the application then uses this identifier to refer to the key so that it can be loaded from memory for encryption operations. Since there can be up to four keys simultaneously activated, data-element encryption provides the infrastructure for tasks that require multiple keys.

When encrypting data for data-element encryption, InterSystems IRIS stores the encryption key's unique identifier with the resulting ciphertext. The unique identifier enables the system to identify the key at decryption time using only the ciphertext itself.

This topic describes:

- [Programmatically Manage Keys](#)
- [Data-Element Encryption Calls](#)
- [Support for Re-Encrypting Data in Real Time](#)

Programmatically Manage Keys

Since data-element encryption is available through an API, there are also a set of calls for managing keys:

- `$SYSTEM.Encryption.CreateEncryptionKey`
- `$SYSTEM.Encryption.ActivateEncryptionKey`
- `$SYSTEM.Encryption.DeactivateEncryptionKey`
- `$SYSTEM.Encryption.ListEncryptionKeys`

These are all methods of the `%SYSTEM.Encryption` class.

Data-Element Encryption Calls

The system methods available for data-element encryption are all methods of the `%SYSTEM.Encryption` class and are:

- `$SYSTEM.Encryption.AESCBCManagedKeyEncrypt`
- `$SYSTEM.Encryption.AESCBCManagedKeyDecrypt`
- `$SYSTEM.Encryption.AESCBCManagedKeyEncryptStream`
- `$SYSTEM.Encryption.AESCBCManagedKeyDecryptStream`

These method names all begin with “AESCBCManagedKey” because the methods use AES (the Advanced Encryption Standard) in cipher block chaining (CBC) mode and are part of the suite of tools for managed key encryption.

Important: The AESCBC methods that do not include “ManagedKey” in their names are older methods and cannot be used for these purposes.

\$SYSTEM.Encryption.AESCBCManagedKeyEncrypt

The signature of this method as it is usually called is:

```
$SYSTEM.Encryption.AESCBCManagedKeyEncrypt
(
    plaintext As %String,
    keyID As %String,
)
As %String
```

where:

- *plaintext* — The unencrypted text to be encrypted.
- *keyID* — The GUID of the data-encryption key to be used to encrypt plaintext.
- The method returns the encrypted ciphertext.

If the method fails, it throws either the <FUNCTION> or <ILLEGAL VALUE> error. Place calls to this method in a **Try-Catch** loop; for more information on Try-Catch, see [The TRY-CATCH Mechanism](#).

For more details, see the **\$SYSTEM.Encryption.AESCBCManagedKeyEncrypt** class reference content.

\$SYSTEM.Encryption.AESCBCManagedKeyDecrypt

The signature of this method as it is usually called is:

```
$SYSTEM.Encryption.AESCBCManagedKeyDecrypt
(
    ciphertext As %String
)
As %String
```

where:

- *ciphertext* — The encrypted text to be decrypted.
- The method returns the decrypted plaintext.

If the method fails, it throws either the <FUNCTION> or <ILLEGAL VALUE> error. Place calls to this method in a **Try-Catch** loop; for more information on Try-Catch, see [The TRY-CATCH Mechanism](#).

You do not need to include the key ID with this call, as the key ID is associated with the ciphertext to be decrypted.

For more details, see the **\$SYSTEM.Encryption.AESCBCManagedKeyDecrypt** class reference content.

\$SYSTEM.Encryption.AESCBCManagedKeyEncryptStream

The signature of this method as it is usually called is:

```
$SYSTEM.Encryption.AESCBCManagedKeyEncryptStream
(
    plaintext As %Stream.Object,
    ciphertext As %Stream.Object,
    keyID As %String,
)
As %Status
```

where:

- *plaintext* — The unencrypted stream to be encrypted.
- *ciphertext* — The variable to receive the encrypted stream.

- *keyID* — The GUID of the data-encryption key to be used to encrypt *plaintext*.
- The method returns a %Status code.

For more details, see the **\$SYSTEM.Encryption.AESCBCManagedKeyEncryptStream** class reference content.

\$SYSTEM.Encryption.AESCBCManagedKeyDecryptStream

The signature of this method as it is usually called is:

```
$SYSTEM.Encryption.AESCBCManagedKeyDecryptStream
(
    ciphertext As %Stream.Object,
    plaintext As %Stream.Object
)
As %Status
```

where:

- *ciphertext* — The encrypted stream to be decrypted.
- *plaintext* — The variable to receive the unencrypted stream.
- The method returns a %Status code.

You do not need to include the key ID with this call, as the key ID is associated with the ciphertext to be decrypted.

For more details, see the **\$SYSTEM.Encryption.AESCBCManagedKeyDecryptStream** class reference content.

Support for Re-Encrypting Data in Real Time

Data-element encryption allows InterSystems IRIS applications to support re-encrypting an encrypted data element with a new key.

Because an encrypted data element has its encrypting key identifier stored with it, this simplifies the process of re-encrypting data. Given merely the handle to ciphertext and an activated key, an application can perform re-encryption. For example, data-element encryption supports the ability to re-encrypt sensitive data without any downtime; this is particularly useful for users required to perform this action for legal reasons, such as those fulfilling PCI DSS (Payment Card Industry Data Security Standard) requirements.

If you need to re-encrypt data, create a new key and specify to your application that this is the new encryption key. You can then perform an action such as running a background application to decrypt the elements and encrypt them with the new key. This uses the specified key for encryption and always uses the correct key for decryption, since it is stored with the encrypted data.

Protecting Against Data Loss

Protecting Against Data Loss

To ensure that encrypted data is always available, InterSystems strongly suggests that you take the following preventative actions:

- If you are using key files, then, for each key that you are using:
 1. Create an additional administrator for the key file.
 2. Record the username and password of that administrator on paper.
 3. Place the recorded username and password in a physically secure location, such as a fireproof safe that is sufficiently far from where the key is in use.
 4. Create a backup copy of the key file and place it in the same secure location as the recorded username and password.
- If you are using a KMIP server, back up the contents of that server according to your server vendor's instructions.

CAUTION: Failure to take these precautions can result in a situation where the encrypted data will be *permanently inaccessible* — there will be no way to read it.

Handling Emergency Situations

Handling Emergency Situations

Emergency situations involving encrypted data can result permanently losing access to the encrypted data. If an emergency situation arises, you must act immediately to minimize the risk of the data being lost forever.

This topic describes the steps to take if an emergency situation with encrypted data arises. To take preventative steps against an emergency situation, see [Protecting Against Data Loss](#).

Handle Emergency Situations When Using Key Files

This topic describes what to do under certain circumstances when you are in danger of losing data. These situations include:

- [If the File Containing an Activated Key is Damaged or Missing](#)
 - [If There Is a Backup Copy of the Key File with a Known Administrator Username and Password](#)
 - [If There Is No Backup Copy of the Key File or the Key Has No Known Administrator Username and Password](#)
- CAUTION:** This is a *dire* situation. Act *immediately*.
- [If the Database Encryption Key File Is Required at Startup and Is Not Present](#)
 - [If You Can Make the Key File Available](#)
 - [If a Backup Key File Is Available](#)
 - [If No Key File Is Available](#)

If the File Containing an Activated Key is Damaged or Missing

In this situation, the following circumstances have occurred:

- A database encryption key has been activated for the InterSystems IRIS® data platform instance.
- InterSystems IRIS is using encrypted data.
- The key file containing the database encryption key becomes damaged.

If There Is a Backup Copy of the Key File with a Known Administrator Username and Password

CAUTION: This procedure is for an emergency situation, where encrypted data in InterSystems IRIS databases is in danger of being lost.

If the file containing an activated key becomes inaccessible or damaged, *immediately* perform the following procedure:

1. Get the backup copy of the key file. This is the copy that you stored as described in [Protection from Accidental Loss of Access to Encrypted Data](#).
2. Make a new backup copy of the key file and store it in a safe place.
3. Set up InterSystems IRIS to use the new copy of the key:
 - If you are using interactive startup, incorporate the new copy of the key into your startup procedures.
 - If you are using unattended startup, then reconfigure your startup options using the new copy of the key file — even if you are setting it up for the same options as before.

If There Is No Backup Copy of the Key File or the Key has No Known Administrator Username and Password

WARNING! THIS PROCEDURE IS FOR AN EMERGENCY SITUATION, WHERE ENCRYPTED DATA IN INTERSYSTEMS IRIS DATABASES IS IN DANGER OF BEING LOST.

If the file containing the activated key becomes inaccessible or damaged while InterSystems IRIS is running, *immediately* perform the following procedure for each database encrypted with that key:

1. **WARNING!** Shutting down InterSystems IRIS or deactivating the active key will cause the permanent loss of your data.

Do not shut down InterSystems IRIS.

Do not deactivate the currently active key.

2. Contact the InterSystems Worldwide Response Center. Engineers there can help guide you through the following procedure and answer any questions that may arise.
3. Dismount the database. This prevents all users from making any changes to the database with encrypted content while copying its data to an unencrypted database:
 - a. From the Management Portal home page, go to the **Databases** page (**System Operation > Databases**).
 - b. On the **Databases** page, if the encrypted database is mounted, select the **Dismount** option in the next-to-last column in that database's row. Then select **OK** in the confirmation dialog.
 - c. When the **Databases** page appears again, select the **Mount** option in the last column in the database's row.
 - d. On the **Mount database** confirmation screen, check the **Read Only** box and select **OK**.

It is critical that no one makes any changes to the database during this procedure. Mounting the database read-only prevents any user from changing any data.

4. Copy all data in unencrypted form to another database. The procedure for copying the data is:

- a. In the Terminal, go to the %SYS namespace:

```
REGULARNAMESPACE>set $namespace="%SYS"
```

- b. From that namespace, run the **^GBLOCKCOPY** command:

```
%SYS>d ^GBLOCKCOPY
```

This routine will do a fast global copy from a database to another database or to a namespace. If a namespace is the destination, the global will follow any mappings set up for the namespace.

```
1) Interactive copy
2) Batch copy
3) Exit
```

```
Option?1
```

- c. At the **^GBLOCKCOPY** prompt, specify 1, for an interactive copy:

```
Option? 1
```

```
1) Copy from Database to Database
2) Copy from Database to Namespace
3) Exit
```

```
Option?
```

- d. When **^GBLOCKCOPY** prompts for a copy type, select 1, for copying from database to database

```
Option? 1
Source Directory for Copy (? for List)?
```

Here, either specify the name of the encrypted database or enter ? to see a numbered list of databases, which includes the encrypted database. If you enter ?, **^GBLOCKCOPY** displays a list such as this one:

```
Source Directory for Copy (? for List)? ?
1) C:\InterSystems\MyIRIS\mgr\
2) C:\InterSystems\MyIRIS\mgr\irislocaldata\
3) C:\InterSystems\MyIRIS\mgr\irisaudit\
4) C:\InterSystems\MyIRIS\mgr\irislib\
5) C:\InterSystems\MyIRIS\mgr\iristemp\
6) C:\InterSystems\MyIRIS\mgr\encrypted1\
7) C:\InterSystems\MyIRIS\mgr\encrypted2\
8) C:\InterSystems\MyIRIS\mgr\unencrypted\

Source Directory for Copy (? for List)?
```

Enter the number of the encrypted database, such as 7 here.

- e. When **^GBLOCKCOPY** prompts for a destination directory for copying the data, enter the name of an unencrypted database or ? for a list similar to the one for the source directory.
- f. When **^GBLOCKCOPY** asks if you wish to copy all globals, enter Yes (can be Yes, Y, y, and so on):

```
All Globals? No => y
```

- g. If there is an empty global in the database, **^GBLOCKCOPY** will now ask if you wish to copy it. This will appear something like the following:

```
All Globals? No => y
^oddBIND      contains no data
Include it anyway? No =>
```

Enter No (can be No, N, n, and so on), which is the default.

- h. **^GBLOCKCOPY** then asks if you wish to skip all the other empty globals. Enter Yes (can be Yes, Y, y, and so on), which is the default:

```
Exclude any other similar globals without asking again? Yes =>
```

There then appears a list of all the empty globals that are not being copied:

```
Exclude any other similar globals without asking again? Yes => Yes
^oddCOM      contains no data -- not included
^oddDEP      contains no data -- not included
^oddEXT      contains no data -- not included
^oddEXTR     contains no data -- not included
^oddMAP      contains no data -- not included
^oddPKG      contains no data -- not included
^oddPROC     contains no data -- not included
^oddPROJECT  contains no data -- not included
^oddSQL      contains no data -- not included
^oddStudioDocument contains no data -- not included
^oddStudioMenu contains no data -- not included
^oddTSQL     contains no data -- not included
^oddXML      contains no data -- not included
^rBACKUP     contains no data -- not included
^rINC        contains no data -- not included
^rINCSAVE    contains no data -- not included
^rINDEXEXT   contains no data -- not included
^rINDEXSQL   contains no data -- not included
^rMACSAVE    contains no data -- not included
9 items selected from
29 available globals
```

- i. **^GBLOCKCOPY** then asks if you wish to disable journaling for this operation:

```
Turn journaling off for this copy? Yes =>
```

Enter Yes (can be Yes, Y, y, and so on), which is the default.

- j. **^GBLOCKCOPY** then asks if to confirm that you wish to copy the data:

```
Confirm copy? Yes =>
```

Enter **Yes** (can be **Yes**, **Y**, **y**, and so on), which is the default. Depending on the size of the database and the speed of the processor, you may see the status of the copy as it progresses. When it completes, **^GBLOCKCOPY** displays a message such as:

```
Copy of data has completed
```

- k. **^GBLOCKCOPY** then asks if you wish to save statistics associated with the copy. Enter **No** (can be **No**, **N**, **n**, and so on), which is the default:

```
Do you want to save statistics for later review? No =>
```

Control then returns to the main prompt.

5. Test that the copied data is valid. You can do this by examining the classes, tables, or globals in the Management Portal's System Explorer for the database into which **^GBLOCKCOPY** has copied the data.
6. If the data is valid, perform steps 3 and 4 of this procedure for each database encrypted with the inaccessible or damaged key.
7. Once you have made copies of every encrypted database into an unencrypted database, make a second copy of each database, preferably to a different machine than that which holds the first copy of each.
8. Now — *and only now* — you can dismount all encrypted databases and deactivate the active key (that is, the key for which the key file is missing or damaged). InterSystems IRIS requires that you dismount all encrypted databases prior to deactivating their key.

You now have your data in one or more unencrypted databases and there is no activated key.

To re-encrypt the formerly encrypted databases, the procedure is:

1. Create a new database encryption key according to the procedure described in [Creating a Key](#).
2. Create a new backup copy of the key file as described in [Protection from Accidental Loss of Access to Encrypted Data](#).

CAUTION: Make sure you take the precautions described in [Protection from Accidental Loss of Access to Encrypted Data](#); failure to follow these procedures can result in the permanent loss of data.

3. Create one or more new encrypted databases, using the new key.
4. Import the data exported in the previous procedure into the new encrypted database(s).

Your data is now stored in encrypted databases for which you have a valid key and a backup copy of the key file containing that key.

If the Database Encryption Key File Is Required at Startup and Is Not Present

Under certain conditions related to the required use of a database encryption key file at startup, the system starts in single-user mode. These conditions are:

- InterSystems IRIS is configured for either interactive or unattended startup.
- Startup specifies that journal files and/or the IRISTEMP and IRISLOCALDATA databases are encrypted, or an encrypted database is specified as required at startup.
- The database encryption key file is not present.

If You Can Make the Key File Available

This situation may have been caused simply by the appropriate key file not being present at InterSystems IRIS startup time — such as if the media holding it is not currently available.

To correct the condition, after InterSystems IRIS starts running in single-user mode, then the procedure is:

1. Shut down InterSystems IRIS. For example, if the instance of InterSystems IRIS is called “MyIRIS”, the command to do this would be:

```
iris force MyIRIS
```

2. If you know the location where InterSystems IRIS is expecting to find the database encryption key file, then place the key file there. (Otherwise, you need to run **^STURECOV** as specified in the next section.)
3. Start InterSystems IRIS again.

InterSystems IRIS should start in its typical mode (multi-user mode) and operate as expected.

If a Backup Key File Is Available

If the appropriate key file is not present at InterSystems IRIS startup time and is not available, you may have a backup key file available. If so, then to correct the condition, after InterSystems IRIS starts running in single-user mode, then the procedure is:

1. Contact InterSystems Worldwide Response Center. Engineers there can help guide you through the following procedure and answer any questions that may arise.
2. Start a [Administrator Terminal Session](#) according to the instructions in the most recent entry in the messages.log file. Typically, this specifies starting a Terminal session with the **-B** flag..

For example, at a Windows command line, for an instance of InterSystems IRIS called “MyIRIS” that is installed in the default location, the command would be:

```
c:\InterSystems\MyIRIS\bin\irisdb -sc:\InterSystems\MyIRIS\mgr -B
```

This connects you with InterSystems IRIS in the operating system terminal window; the prompt in that window changes from the operating system prompt to the InterSystems IRIS **%SYS** prompt.

3. If you have or can obtain a copy of the database encryption key file (such as a backup), then place a copy of the key file in a location accessible to InterSystems IRIS.
4. Run the **^STURECOV** (startup recovery) routine at the Terminal prompt. In that routine, activate the encryption key using an administrator username and password in that file. (You do not need to exit **^STURECOV** when you have completed this process.)
5. When you are satisfied that InterSystems IRIS is ready for use, use **^STURECOV** to complete the startup procedure. InterSystems IRIS then starts in multi-user mode.

InterSystems IRIS should now operate as expected.

If No Key File Is Available

If you do not have any copy of the database encryption key file, then the procedure is:

1. Contact InterSystems Worldwide Response Center (WRC). Engineers there can help guide you through the following procedure and answer any questions that may arise.
2. Start a [Administrator Terminal Session](#) according to the instructions in the most recent entry in the messages.log file. Typically, this specifies starting a Terminal session with the **-B** flag..

For example, at a Windows command line, for an instance of InterSystems IRIS called “MyIRIS” that is installed in the default location, the command would be:

```
c:\InterSystems\MyIRIS\bin\irisdb -sc:\InterSystems\MyIRIS\mgr -B
```

This connects you with InterSystems IRIS in the operating system terminal window; the prompt in that window changes from the operating system prompt to the InterSystems IRIS %SYS prompt.

3. If any encrypted databases require mounting at startup, disable this feature for them:
 - a. From the Management Portal Home page, go to the **Local Databases** page (**System Administration > Configuration > System Configuration > Local Databases**).
 - b. Click the name of the database in the table of databases. This displays the **Edit:** page for the database.
 - c. On the **Edit:** page, clear the **Mount Required at Startup** check box.
 - d. Click **Save**.
4. Run the ^STURECOV routine at the Terminal prompt. In that routine, configure InterSystems IRIS database startup options not to require a database encryption key. This means that the IRISTEMP and IRISLOCALDATA databases as well as journal files should now operate as expected; it also means that any encrypted databases cannot be mounted.
5. When you are satisfied that InterSystems IRIS is ready for use, use ^STURECOV to complete the startup procedure. InterSystems IRIS then starts in multi-user mode.

As you perform this procedure, you may need to perform other actions, according to the instructions of the representative from the WRC. Follow these instructions.

CAUTION: If you have not performed the actions described in [Protection from Accidental Loss of Access to Encrypted Data](#), then your data may no longer be available in any form. This is a very serious problem, but if you do not have a key, there is no way to retrieve the lost data.

Handle Emergency Situations When Using a KMIP Server

This topic describes what to do under certain circumstances when you are using a KMIP server and are in danger of losing data. These situations include:

- [If the KMIP Server Holding an Activated Key is Damaged or Missing](#)
 - [If There Is a Backup Copy of the Key on the KMIP Server](#)
 - [If There Is No Backup Copy of the Key on the KMIP Server](#)

WARNING! This is a *dire* situation. Act *immediately*.

- [If the KMIP Server Is Required at Startup and Is Not Accessible](#)
 - [If the Connection to the KMIP Server is Briefly Unavailable](#)
 - [If the KMIP Server Suffers a Longer-Term Outage](#)

If the KMIP Server Holding an Activated Key is Damaged or Missing

In this situation, the following circumstances have occurred:

- A database encryption key has been activated for the InterSystems IRIS instance
- InterSystems IRIS is using encrypted data
- The KMIP server the database encryption key becomes damaged

If There Is a Backup Copy of the Key on the KMIP Server

If KMIP server holding an activated key becomes inaccessible or damaged, *immediately* perform restore procedures for the KMIP server according to your vendor's instructions.

If There Is No Backup Copy of the Key on the KMIP Server

WARNING! THIS PROCEDURE IS FOR AN EMERGENCY SITUATION, WHERE ENCRYPTED DATA IN INTERSYSTEMS IRIS DATABASES IS IN DANGER OF BEING LOST.

If there is no way to restore the KMIP server holding the activated key from backup, *immediately* perform the following procedure for each database encrypted with that key:

1. **WARNING!** Shutting down InterSystems IRIS or deactivating the active key will cause the permanent loss of your data.

Do not shut down InterSystems IRIS.

Do not deactivate the currently active key.

2. Contact the InterSystems Worldwide Response Center. Engineers there can help guide you through the following procedure and answer any questions that may arise.
3. Dismount the database. This prevents all users from making any changes to the database with encrypted content while copying its data to an unencrypted database:
 - a. From the Management Portal home page, go to the **Databases** page (**System Operation** > **Databases**).
 - b. On the **Databases** page, if the encrypted database is mounted, select the **Dismount** option in the next-to-last column in that database's row. Then select **OK** in the confirmation dialog.
 - c. When the **Databases** page appears again, select the **Mount** option in the last column in the database's row.
 - d. On the **Mount database** confirmation screen, check the **Read Only** box and select **OK**.

It is critical that no one makes any changes to the database during this procedure. Mounting the database read-only prevents any user from changing any data.

4. Copy all data in unencrypted form to another database. The procedure for copying the data is:
 - a. In the Terminal, go to the %SYS namespace:

```
REGULARNAMESPACE>set $namespace="%SYS"
```

- b. From that namespace, run the **^GBLOCKCOPY** command:

```
%SYS>do ^GBLOCKCOPY
```

```
This routine will do a fast global copy from a database to another database or  
to a namespace. If a namespace is the destination, the global will follow any  
mappings set up for the namespace.
```

```
1) Interactive copy  
2) Batch copy  
3) Exit
```

```
Option?1
```

- c. At the **^GBLOCKCOPY** prompt, specify 1, for an interactive copy:

```
Option? 1
```

```
1) Copy from Database to Database  
2) Copy from Database to Namespace  
3) Exit
```

```
Option?
```

- d. When **^GBLOCKCOPY** prompts for a copy type, select 1, for copying from database to database

```
Option? 1
Source Directory for Copy (? for List)?
```

Here, either specify the name of the encrypted database or enter ? to see a numbered list of databases, which includes the encrypted database. If you enter ?, **^GBLOCKCOPY** displays a list such as this one:

```
Source Directory for Copy (? for List)? ?

1) C:\InterSystems\MyIRIS\mgr\
2) C:\InterSystems\MyIRIS\mgr\irislocaldata\
3) C:\InterSystems\MyIRIS\mgr\irisaudit\
4) C:\InterSystems\MyIRIS\mgr\irislib\
5) C:\InterSystems\MyIRIS\mgr\iristemp\
6) C:\InterSystems\MyIRIS\mgr\encrypted1\
7) C:\InterSystems\MyIRIS\mgr\encrypted2\
8) C:\InterSystems\MyIRIS\mgr\unencrypted\

Source Directory for Copy (? for List)?
```

Enter the number of the encrypted database, such as 7 here.

- e. When **^GBLOCKCOPY** prompts for a destination directory for copying the data, enter the name of an unencrypted database or ? for a list similar to the one for the source directory.
- f. When **^GBLOCKCOPY** asks if you wish to copy all globals, enter Yes (can be Yes, Y, y, and so on):

```
All Globals? No => y
```

- g. If there is an empty global in the database, **^GBLOCKCOPY** will now ask if you wish to copy it. This will appear something like the following:

```
All Globals? No => y

^oddBIND      contains no data
Include it anyway? No =>
```

Enter No (can be No, N, n, and so on), which is the default.

- h. **^GBLOCKCOPY** then asks if you wish to skip all the other empty globals. Enter Yes (can be Yes, Y, y, and so on), which is the default:

```
Exclude any other similar globals without asking again? Yes =>
```

There then appears a list of all the empty globals that are not being copied:

```
Exclude any other similar globals without asking again? Yes => Yes
^oddCOM      contains no data -- not included
^oddDEP      contains no data -- not included
^oddEXT      contains no data -- not included
^oddEXTR     contains no data -- not included
^oddMAP      contains no data -- not included
^oddPKG      contains no data -- not included
^oddPROC     contains no data -- not included
^oddPROJECT  contains no data -- not included
^oddSQL      contains no data -- not included
^oddStudioDocument contains no data -- not included
^oddStudioMenu contains no data -- not included
^oddTSQL     contains no data -- not included
^oddXML      contains no data -- not included
^rBACKUP     contains no data -- not included
^rINC        contains no data -- not included
^rINCSAVE    contains no data -- not included
^rINDEXEXT   contains no data -- not included
^rINDEXSQL   contains no data -- not included
^rMACSAVE    contains no data -- not included
9 items selected from
29 available globals
```

- i. **^GBLOCKCOPY** then asks if you wish to disable journaling for this operation:

Turn journaling off for this copy? Yes =>

Enter Yes (can be Yes, Y, y, and so on), which is the default.

- j. **^GBLOCKCOPY** then asks if to confirm that you wish to copy the data:

Confirm copy? Yes =>

Enter Yes (can be Yes, Y, y, and so on), which is the default. Depending on the size of the database and the speed of the processor, you may see the status of the copy as it progresses. When it completes, **^GBLOCKCOPY** displays a message such as:

Copy of data has completed

- k. **^GBLOCKCOPY** then asks if you wish to save statistics associated with the copy. Enter No (can be No, N, n, and so on), which is the default:

Do you want to save statistics for later review? No =>

Control then returns to the main prompt.

5. Test that the copied data is valid. You can do this by examining the classes, tables, or globals in the Management Portal's System Explorer for the database into which **^GBLOCKCOPY** has copied the data.
6. If the data is valid, perform steps 3 and 4 of this procedure for each database encrypted with the inaccessible or damaged key.
7. Once you have made copies of every encrypted database into an unencrypted database, make a second copy of each database, preferably to a different machine than that which holds the first copy of each.
8. Now — *and only now* — you can dismount all encrypted databases and deactivate the active key (that is, the key for which the key file is missing or damaged). InterSystems IRIS requires that you dismount all encrypted databases prior to deactivating their key.

You now have your data in one or more unencrypted databases and there is no activated key.

To re-encrypt the formerly encrypted databases, the procedure is:

1. Create a new database encryption key according to the procedure described in [Create a Key on the KMIP Server](#).
2. Create a new backup copy of the key file as described in [Protection from Accidental Loss of Access to Encrypted Data](#).

CAUTION: Make sure you take the precautions described in [Protection from Accidental Loss of Access to Encrypted Data](#); failure to follow these procedures can result in the permanent loss of data.

3. Create one or more new encrypted databases, using the new key.
4. Import the data exported in the previous procedure into the new encrypted database(s).

Your data is now stored in encrypted databases for which you have a valid key and a backup copy of the key file containing that key.

If the KMIP Server Is Required at Startup and Is Not Accessible

Under certain conditions related to the required use of one or more database encryption keys at startup, the system starts in single-user mode. These conditions are:

- InterSystems IRIS is configured for either interactive or unattended startup.

- Startup specifies that journal files and/or the IRISTEMP and IRISLOCALDATA databases are encrypted, or an encrypted database is specified as required at startup.
- The KMIP server that holds the required database encryption key is not accessible.

If the Connection to the KMIP Server is Briefly Unavailable

The simplest solution to this case is when there has been a problem such as a network outage or the KMIP server is otherwise temporarily not running; in these cases, address networking or server problem and, if required, restart InterSystems IRIS.

If the KMIP Server Suffers a Longer-Term Outage

If it is not possible to connect to the KMIP server longer-term:

1. Start a [Administrator Terminal Session](#) according to the instructions in the most recent entry in the messages.log file. Typically, this specifies starting a Terminal session with the `-B` flag..

For example, at a Windows command line, for an instance of InterSystems IRIS called “MyIRIS” that is installed in the default location, the command would be:

```
c:\InterSystems\MyIRIS\bin\irisdb -sc:\InterSystems\MyIRIS\mgr -B
```

This connects you with InterSystems IRIS in the operating system terminal window; the prompt in that window changes from the operating system prompt to the InterSystems IRIS `%SYS` prompt.

2. If any encrypted databases require mounting at startup, disable this feature for them:
 - a. From the Management Portal Home page, go to the **Local Databases** page (**System Administration > Configuration > System Configuration > Local Databases**).
 - b. Click the name of the database in the table of databases. This displays the **Edit:** page for the database.
 - c. On the **Edit:** page, clear the **Mount Required at Startup** check box.
 - d. Click **Save**.
3. Run the `^STURECOV` routine at the Terminal prompt. In that routine, configure InterSystems IRIS database startup options not to require a database encryption key. This means that the IRISTEMP and IRISLOCALDATA databases as well as journal files should now operate as expected; it also means that any encrypted databases cannot be mounted.
4. When you are satisfied that InterSystems IRIS is ready for use, use `^STURECOV` to complete the startup procedure. InterSystems IRIS then starts in multi-user mode.

CAUTION: If you have not performed the actions described in [Protection from Accidental Loss of Access to Encrypted Data](#), then your data may no longer be available in any form. This is a very serious problem, but, if you do not have a key, there is no way to retrieve the lost data.

Additional Encryption Information

Additional Encryption Information

This topic addresses additional information about InterSystems IRIS® data platform encryption.

Key File Encryption Information

Database encryption administrator names are stored in the clear in the key file. Database encryption administrator passwords are not stored; when entered, they are used, along with other data, to derive key-encryption keys. If someone can successfully guess a valid password, the password policy is too weak. Key-encryption keys are derived using the PBKDF2 algorithm with 512 bits of salt and 65,536 iterations, making dictionary and brute force attacks impractical.

Encryption and Database-Related Facilities

InterSystems IRIS database encryption protects database files themselves. Regarding related facilities in InterSystems IRIS:

- InterSystems IRIS online backups are not encrypted. To ensure that the InterSystems IRIS database is encrypted in a backup, it is recommended that you quiesce InterSystems IRIS and then perform a file system backup (as described in [External Backup](#)).
- In the write image journal (WIJ) file, the blocks for encrypted databases are encrypted.
- The IRISTEMP and IRISLOCALDATA databases can optionally be encrypted. To provide encryption for IRISTEMP and IRISLOCALDATA, see [Configure Encryption Startup Settings](#).
- You can optionally encrypt journal files; see [Configuring Database Encryption Settings](#).

About Calls to Perform Encryption, Hashing, and Other Key-Related Operations

InterSystems IRIS allows you to perform actions related to data encryption, Base64 encoding, hashing, and generating message authentication codes using various methods of the %SYSTEM.Encryption class. It includes methods that invoke AES encryption, various RSA algorithms, SHA-256 hash functions, and more. Some of the calls include:

- `$System.Encryption.AESCBCManagedKeyEncrypt` and `$System.Encryption.AESCBCManagedKeyDecrypt`
- `$System.Encryption.AESKeyWrap` and `$System.Encryption.AESKeyUnwrap`
- `$System.Encryption.Base64Encode` and `$System.Encryption.Base64Decode`
- `$System.Encryption.RSASHASign` and `$System.Encryption.RSASHAVerify`
- `$System.Encryption.RSAEncrypt` and `$System.Encryption.RSADecrypt`
- `$System.Encryption.SHAHash`

An Example of Using RSAEncrypt and RSADecrypt

Below is an example of using the **RSAEncrypt** and **RSADecrypt** calls. It assumes that:

- The code is running on Windows.
- There is an available certificate, private key, and certificate authority (CA) certificate. (To try this example, you will need to obtain these.)
- All three of these items are in the C:\Keys\ directory.

See the comments within the example for more details of its operations.

ObjectScript

```

set dir = "C:\Keys\"

// certificate for the instance performing encryption and decryption
// and private key associated with that above certificate
set cert = dir_"test.crt"
set key = dir_"test.key"

// certificate for the CA of the instance
set cacert=dir_"ca.crt"

set data = "data to be encrypted"

// create a local set of X.509 credentials with the
// certificate and private key
set credentials = ##class(%SYS.X509Credentials).%New()
set credentials.Alias="TestCreds"
write credentials.LoadCertificate(cert)
write credentials.LoadPrivateKey(key)
write credentials.Save(),!

// encrypt the data using the public key in the certificate, write it
// to the display, and display error information, if there is any
set ciphertext=$System.Encryption.RSAEncrypt(data,credentials.Certificate,cacert)
write ciphertext,!
write $System.Encryption.RSASHA1GetLastError()

// decrypt the data using the private key, write it to the display,
// and display error information, if there is any
write "now decrypting -----",!
set cleartext=$System.Encryption.RSADecrypt(ciphertext,credentials.PrivateKey)
write cleartext,!
write $System.Encryption.RSASHA1GetLastError()

```

FIPS-2 Compliance

FIPS 140–2 Compliance for Database Encryption

On specific platforms, InterSystems IRIS® data platform supports FIPS 140–2 compliant cryptography for database encryption. (FIPS 140–2 refers to Federal Information Processing Standard Publication 140-2, which is available at <https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf>.)

This version of InterSystems IRIS supports FIPS 140–2–compliant cryptography for database encryption on Red Hat Enterprise Linux 8 for x86-64. Red Hat has certificates of validation for the OpenSSL libcrypto.so and libssl.so libraries. When running in FIPS mode, InterSystems IRIS uses these certified libraries. To determine if a minor version of Red Hat Linux has current certification, consult the [Red Hat documentation](#).

Note: With FIPS mode enabled:

- Red Hat 8 supports only TLSv1.2 and TLSv1.3.

For information about Red Hat support for government standards, see <https://access.redhat.com/articles/2918071>.

Important: InterSystems IRIS does not currently support FIPS mode on Red Hat 9.

Enabling FIPS Support

To enable InterSystems IRIS support for FIPS 140–2 compliant cryptography for database encryption, do the following:

1. Download and install the openssl package from the RedHat repository (rhel-8-server-rpms).
2. Enable FIPS mode for the operating system. For these instructions, see the article [How can I make RHEL 6/7/8 FIPS 140-2 compliant?](#) on the Red Hat web site. (Access to this article requires Red Hat login credentials.)
3. Check the directory /usr/lib64 for the following symbolic links. If these do not exist, create them:
 - The symbolic link libssl.so.1.1 should point to the appropriate file (such as libssl.so.1.1.1g), in the same directory.
 - The symbolic link libcrypto.so.1.1 should point to the appropriate file (such as libcrypto.so.1.1.1g), in the same directory.
4. In InterSystems IRIS, specify the **FIPSMODE** CPF parameter as **True** (1). To do so:
 - a. Open the Management Portal.
 - b. Select **System Administration > Configuration > Additional Settings > Startup**.
Here you will see a row for **FIPSMODE**.
 - c. Specify the value for **FIPSMODE** as **True** and save your change.
5. Restart InterSystems IRIS.
6. Enable and configure encrypted databases as outlined in [Using Encrypted Databases](#).

Startup Behavior and messages.log

When InterSystems IRIS is started:

- If **FIPSMODE** is 0, InterSystems IRIS native cryptography is used, including optimized assembly code using Intel AES-NI hardware instructions, if supported by the CPU. In this mode, InterSystems IRIS writes the following to messages.log upon startup:

Terminal

```
FIPS 140-2 compliant cryptography for database encryption is not configured in iris.cpf
```

- If **FIPSMODE** is 1, InterSystems IRIS attempts to resolve references to functions in the /usr/lib64/libcrypto.so FIPS-validated library, and then attempts to initialize the library in FIPS mode. If these steps are successful, InterSystems IRIS writes the following to messages.log:

Terminal

```
FIPS 140-2 compliant cryptography for database encryption is enabled for this instance.
```

- If **FIPSMODE** is 1, but the initialization of the library is unsuccessful, InterSystems IRIS does not start. In this case, messages.log contains the following message:

Terminal

```
FIPS 140-2 compliant cryptography for database encryption initialization failed. Aborting.
```

- On platforms other than lnxrhx64, if **FIPSMODE** is 1, InterSystems IRIS native cryptography is used, and InterSystems IRIS writes the following to messages.log:

Terminal

```
FIPS 140-2 compliant cryptography for database encryption is not supported on this platform.
```

Cryptographic Standards and RFCs

Cryptographic Standards and RFCs

The following are standards and RFCs (requests for comment) that define the cryptographic primitives and algorithms used in InterSystems security:

- AES (Advanced Encryption Standard) encryption — FIPS (Federal Information Processing Standards) 197
- AES Key Wrap —
 - NIST (National Institute of Standards and Technology) document “Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping” (https://csrc.nist.gov/CryptoToolkit/kms/AES_key_wrap.pdf)
 - IETF (Internet Engineering Task Force) RFC 3394
- Base64 encoding — RFC 3548
- Block padding — PKCS (Public-Key Cryptography Standards) #7 and RFC 2040
- CBC (Cipher Block Chaining) cipher mode — NIST 800-38A
- Deterministic random number generator —
 - FIPS PUB 140-2, Annex C
 - FIPS PUB 186-2, Change Notice 1, Appendix 3.1 and Appendix 3.3
- GSS (Generic Security Services) API —
 - The Kerberos Version 5 GSS-API Mechanism — RFC 1964
 - Generic Security Service Application Program Interface, Version 2, Update 1 — RFC 2743
 - Generic Security Service API Version 2: C Bindings — RFC 2744
 - Generic Security Service API Version 2: Java Bindings — RFC 2853
- Kerberos Network Authentication Service (V5) — RFC 1510
- Hash-based Message Authentication Code (HMAC) — FIPS 198 and RFC 2104
- Message Digest 5 (MD5) hash — RFC 1321
- Password-Based Key Derivation Function 2 (PBKDF2) — PKCS #5 v2.1 and RFC 8018
- Secure Hash Algorithm (SHA-1) — FIPS 180-2 and RFC 3174
- Secure Hash Algorithm (SHA-512) — FIPS 180-2 and RFC 6234

All these documents are available online:

- [FIPS documents](#)
- [NIST documents](#)
- [RFCs \(IETF\)](#)

Public Key Infrastructure

The InterSystems Public Key Infrastructure

Important: The InterSystems PKI is for testing purposes only. Do *not* use it in a production setting.

This document covers the following topics:

- [About the InterSystems Public Key Infrastructure \(PKI\)](#)
- [Certificate Authority Server Tasks](#)
 - [Configuring an InterSystems IRIS® Instance as a Certificate Authority Server](#)
 - [Managing Pending Certificate Signing Requests](#)
- [Certificate Authority Client Tasks](#)
 - [Configuring an InterSystems IRIS Instance as a Certificate Authority Client](#)
 - [Submitting a Certificate Signing Request to a Certificate Authority Server](#)
 - [Getting Certificate\(s\) from the Certificate Authority Server](#)

About the InterSystems Public Key Infrastructure (PKI)

A Public Key Infrastructure (PKI) provides a means of creating and managing private keys, public keys, and certificates. These are used for cryptographic operations including encryption, decryption, and digital signing and signature verification. Certificates provide a means of associating a public key with an identity. To do this, a PKI includes a trusted third party known as a certificate authority (CA).

The InterSystems PKI implementation gives InterSystems IRIS the ability to serve as a Certificate Authority (CA). An instance of InterSystems IRIS acting as a CA is known as a CA server; an instance of InterSystems IRIS using a CA's services is known as a CA client. A single instance of InterSystems IRIS can be both a CA server and a CA client. As a CA server, an instance can either generate and use a self-signed CA Certificate, or it can use a CA Certificate issued by a commercial third party or product. As a CA client, an instance is associated with a CA server; the CA client's certificate is available for use with TLS, XML encryption, and signature verification; there is also the option of configuring a CA client to serve as an intermediate CA. Communications involving the PKI occur through web services.

When establishing itself as a CA server, an instance of InterSystems IRIS either creates a public/private key pair and then embeds the public key in a self-signed X.509 certificate or it uses a private key and X.509 certificate signed by an outside CA. X.509 is an industry-standard certificate structure that associates a public key with both identifying information for an entity and other related data; this identifying information is known as a subject Distinguished Name (DN), and consists of various specific information regarding an entity's organization, location, or both. You can use X.509 certificates to provide a high level of public-key-based security if, and only if, appropriate security policies regarding the protection of private keys and the issuance of certificates are enforced, including strict control of the CA server's private key file, preferably stored on removable media which can be physically secured when not in use.

To use the InterSystems IRIS PKI infrastructure from the Management Portal, all actions start from the **Public Key Infrastructure** page (**System Administration > Security > Public Key Infrastructure**).

For more background on PKI and CAs, see “[About Public Key Infrastructure \(PKI\)](#).” For technical details about the TLS calls underlying the InterSystems PKI, see the [OpenSSL](#) library. For technical details about X.509 certificates, see [RFC 5280](#), “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.”

Help for Management Portal PKI Tasks

The following are links to help for [PKI](#) tasks:

- Certificate Authority Client
 - [Submit Certificate Signing Request to Certificate Authority server](#)
 - [Get Certificate\(s\) from Certificate Authority server](#)
 - [Configure local Certificate Authority client](#)
- Certificate Authority Server
 - [Process pending Certificate Signing Requests](#)
 - [Configure local Certificate Authority server](#)

Certificate Authority Server Tasks

An InterSystems IRIS instance can serve as a certificate authority (CA) server. This involves:

1. [Configuring an InterSystems IRIS instance as a CA server](#). This involves providing information that is either related to the certificate or that the CA server uses for processing certificate signing requests.
2. [Managing pending certificate signing requests \(CSRs\)](#). This is the ongoing work of a CA server.

Note: Because these tasks are for the CA server administrator, this section is addressed to those administrators. This differs from the tasks in the CA client tasks, which are addressed to CA client administrators/technical contacts.

Configure an InterSystems IRIS Instance as a Certificate Authority Server

Before any PKI operations are possible, you need to configure an InterSystems IRIS instance as a Certificate Authority (CA) server. This involves either:

- [Configuring a CA Server with a New Private Key and Certificate](#)
- [Configuring a CA Server with an Existing Private Key and Certificate](#)

It may also involve [reinitializing a CA server](#).

Configure a CA Server with a New Private Key and Certificate

If you are creating a new private key and certificate, the procedure is:

1. For the selected InterSystems IRIS instance, in the Management Portal, go to the **Public Key Infrastructure** page (**System Administration > Security > Public Key Infrastructure**).
2. On the **Public Key Infrastructure** page, under **Certificate Authority Server**, select **Configure Local Certificate Authority server**. This displays fields for (1) the file name root for the CA server's certificate and private key and (2) the directory that holds these files.

Important: If you specify a path and file name root that point to an existing certificate and private key, InterSystems IRIS uses these for the CA server. Otherwise, it creates a new certificate and private key. (Also, if only one of the files exists, InterSystems IRIS renames it by appending the `.old` suffix to it and creates new files.)

The fields are:

- **File name root for Certificate Authority's Certificate and Private Key files (without extension)** — Required. Specifies the part of the name of the private key and certificate files that is common to each. This can be for an existing pair of files or for a new pair of files. Hence, if you select `MyCA` as the file name root, the private key is stored in the `MyCA.key` file and the certificate is stored in the `MyCA.cer` file. Valid characters for this field are alphanumeric characters, the hyphen, and the underscore. The root cannot be the string “cache”.

- **Directory for Certificate Authority's Certificate and Private Key files** — Required. The path to a directory for storing the CA's certificate and private key files; if the directory does not exist, InterSystems IRIS attempts to create it. This directory should always be on an external device (not a local hard drive or a network server), preferably on an encrypted external device. As this is the directory that holds the CA's private key, it is extremely important that this be a completely secure location. If you provide a relative path here, the path is relative to <install-dir>/mgr/ for the InterSystems IRIS instance.
3. Click **Next** to continue.
 4. The fields that appear next depend on whether you are creating a new private key and certificate pair or using an existing private key and certificate. When you are creating a new private key and certificate, InterSystems IRIS displays the following fields:
 - **Password to Certificate Authority's Private Key file and Confirm Password** — Required. The password to encrypt and decrypt the CA's private key file.
 - **Certificate Authority Subject Distinguished Name** — The set of one or more name-value pairs that define the distinguished name (DN) that describes the bearer of the CA certificate. You must provide a value for at least one attribute. The attributes are:
 - **Country** — A two-letter code identifying the country, using the [ISO country codes](#).
 - **State or Province** — The name of the CA's state or province. The convention is not to use this field for CA certificates.
 - **Locality** — The name of the CA's municipality. The convention is not to use this field for CA certificates.
 - **Organization** — Name of the organization that is administering the CA. By convention, this value is spelled out in full, such as "InterSystems Corporation," rather than simply "InterSystems" or "InterSystems Corp."
 - **Organizational Unit** — Any other organizational information or special commentary on the CA. Examples of this can include the CA's department, a statement that the CA is for use only within an enterprise, and so on.
 - **Common Name** — A descriptive name for the CA, such as "Documentation Test CA."
 - **Validity period for Certificate Authority's Certificate (days)** — Required. The validity period (lifespan) for the CA certificate itself.
 - **Validity period for Certificates issued by Certificate Authority (days)** — Required. The validity period (lifespan) for certificates that the CA issues for its clients.
 - **Configure email** — Information required for the email account for managing the CA and its tasks.
 - **SMTP server** — The Simple Mail Transfer Protocol (SMTP) server that handles the CA mail in the form of a fully qualified host name, such as "MyMachine.MyDomain.com".
 - **SMTP username** — A username that can be authenticated by the specified SMTP server. This field does not require a fully qualified username.
 - **SMTP password** — The password associated with the SMTP username.
 - **Confirm password** — A confirmation of the password associated with the SMTP username.
 - **Certificate Authority server administrator's email address** — The user who receives certificate signing requests for the CA. This field requires a fully qualified username, such as "CAMgr@MyDomain.com".
 5. Complete these fields as required and click **Save**. InterSystems IRIS displays a message indicating success, such as:

```
Certificate Authority server successfully configured.
Created new files: C:\pki\FileNameRoot.cer .key, and .srl.
Certificate Authority Certificate SHA-1 fingerprint:
E3:FB:30:09:53:90:9A:31:30:D3:F0:07:8F:64:65:CD:11:0A:1A:A2
Confirmation email sent to: CAserver-admin@intersystems.com
```

This indicates that a private key, certificate, and their associated SRL (serial) file have been created. (Otherwise, InterSystems IRIS displays an error message.)

Once the files have been created, it is *strongly* recommended that you store the private key on removable media that can be physically secured.

WARNING! It is critical that you properly protect all private keys, and most important that you protect the private key of a CA. The exposure of a private key can result in security breaches, data exposure, financial losses, and legal vulnerability.

If it has succeeded, InterSystems IRIS has performed the following actions:

- Creating a key pair.
- Saving the private key to a file to a specified location with a specified file name root (see below).
- Creating a self-signed CA certificate containing the public key.
- Saving the certificate to a file to a specified location with a specified file name root (see below).
- Creating a counter of the number of certificates issued and storing it in an SRL (serial) file in the same directory as the certificate and the private key. (Each time the CA issues a new certificate, InterSystems IRIS gives the certificate a unique serial number based on this counter and then increments the value in the SRL file.)

Once you have created the CA private key and certificate, you can process certificate signing requests (CSRs). When a CA client creates a CSR, you, as CA administrator, will receive email notification about this.

Configure a CA Server with an Existing Private Key and Certificate

If you are using an existing private key and certificate (such as from another InterSystems IRIS CA, or from an external CA, such as a commercial CA), the procedure is:

1. Create or obtain a private key and certificate. The certificate must be in PEM format, or you must be able to convert it to PEM format.
2. If they do not already have identical file name roots, rename them as *filenameroot.cer* for the certificate and *filenameroot.key* for the private key, where *filenameroot* is the file name root you wish to use.
3. Store both files in the same directory, making sure that this directory is accessible to the InterSystems IRIS instance that you are configuring as a CA server. This directory should always be on an external device (not a local hard drive or a network server), preferably on an encrypted external device. As this is the directory that holds the CA's private key, it is extremely important that this be a completely secure location.

WARNING! It is critical that you properly protect all private keys, and most important that you protect the private key of a CA. The exposure of a private key can result in security breaches, data exposure, financial losses, and legal vulnerability.

4. For the selected InterSystems IRIS instance, in the Management Portal, go to the **Public Key Infrastructure** page (**System Administration > Security > Public Key Infrastructure**).
5. On the **Public Key Infrastructure** page, under **Certificate Authority Server**, select **Configure Local Certificate Authority server**. Complete the fields on this page as follows:
 - **File name root for Certificate Authority's Certificate and Private Key files (without extension)** — Required. The part of the name of the private key and certificate files that is common to each. For this value, use the file name root that the files have or that you selected in step 2 of this procedure.
 - **Directory for Certificate Authority's Certificate and Private Key files** — Required. The path to a directory that holds the CA's certificate and private key files. For this value, use the directory that you selected in step 3. If you provide a relative path here, the path is relative to <install-dir>/mgr/ for the InterSystems IRIS instance.

6. Click **Next** to continue.
7. The fields that appear next depend on whether you are creating a new private key and certificate pair or using an existing private key and certificate. When you are using an existing private key and certificate, InterSystems IRIS displays the following fields:
 - **Validity period for Certificates issued by Certificate Authority (days)** — Required. The validity period (lifespan) for certificates that the CA issues for its clients.
 - **Configure email** — Information required for the email account for managing the CA and its tasks.
 - **SMTP server** — The Simple Mail Transfer Protocol (SMTP) server that handles the CA mail in the form of a fully qualified host name, such as “MyMachine.MyDomain.com”.
 - **SMTP username** — A username that can be authenticated by the specified SMTP server. This field does not require a fully qualified username.
 - **SMTP password** — The password associated with the SMTP username.
 - **Confirm password** — A confirmation of the password associated with the SMTP username.
 - **Certificate Authority server administrator's email address** — The user who receives certificate signing requests for the CA. This field requires a fully qualified username, such as “CAMgr@MyDomain.com”.

Important: If the Management Portal displays more fields than these, then you have not properly directed it to the private key and certificate that you wish to use. If you complete all the displayed fields, click **Save**, and there is success, InterSystems IRIS will have created a new private key and certificate for the CA server.

8. Click **Save**. When you save the configuration information for the local CA server, InterSystems IRIS uses the existing certificate and private key. (It will also create an SRL file, if one does not exist.) It will display a success message such as:

```
Certificate Authority server successfully configured.
Using existing files: C:\pki\FileNameRoot.cer and .key
Certificate Authority Certificate SHA-1 fingerprint:
E3:FB:30:09:53:90:9A:31:30:D3:F0:07:8F:64:65:CD:11:0A:1A:A2
Confirmation email sent to: CAserver-admin@intersystems.com
```

As with creating a new private key and certificate, at this point, the CA server is configured and is now ready to process certificate signing requests (CSRs). Again, when a CA client creates a CSR, you, as CA administrator, will receive email notification about this.

Reinitialize a CA Server

If you have already configured an instance as a CA server, then there is a **Reinitialize** button on the page for configuring a CA. Selecting it has the following effects:

- It deletes all configuration information for the CA server.
- It discards all information for issued certificates.
- It discards all certificate signing requests pending with the CA.

Note: Reinitialization does not delete the files containing the private key or existing certificate for the CA, nor does it delete the CA's existing SRL file; in fact, these are still valid and can be used. Also, it does not render any already signed certificates invalid.

When you click the button, there is a prompt to confirm that you want to reinitialize the CA. After reinitialization, you can configure a new CA server.

Manage Pending Certificate Signing Requests

Once the Certificate Authority (CA) server has been configured, the principal task associated with the CA server is managing certificate signing requests (CSRs) from potential CA clients. This can involve:

- [Processing a Certificate Signing Request \(CSR\)](#)
- [Deleting a Certificate Signing Request \(CSR\)](#)

If processing leads to approving the request, the CA server issues an X.509 certificate signed with the CA's private key, and sends email notification of the issued certificate's serial number to the CA client's technical contact. It is also possible to delete (that is, reject) a request.

A critical step in this process is *verification*, in which the CA administrator uses communications that prevent impersonation to verify the identity of the requester, the authority of the technical contact to hold a certificate with the requested DN, and the purpose for which the certificate is being issued. (To do this, the CA server's administrator uses the contact information received from the potential CA client along with the CSR.)

Process a Certificate Signing Request (CSR)

To process a request is to convert the CSR into a certificate. The procedure is:

1. In the Management Portal, go to the **Public Key Infrastructure** page (**System Administration > Security > Public Key Infrastructure**).
2. On the **Public Key Infrastructure** page, under **Certificate Authority Server**, select **Process pending Certificate Signing Requests**. This displays a table of CSRs that the CA has received and not processed or deleted; to the right of each CSR there are **Process** and **Delete** links.
3. Mount the media containing the CA server certificate and private key files. (This is the media on which you stored these files while [configuring an InterSystems IRIS instance as a CA server](#).)
4. To process a CSR, click **Process**. This displays the contents of the CSR.
5. Prior to issuing the certificate, you need to specify the use of the certificate. The choices for the **Certificate Usage** radio buttons specify which operations the certificate can perform:
 - **TLS/SSL and XML security** — For CA clients that are directly using various security capabilities within InterSystems IRIS.
 - **Intermediate Certificate Authority** — For CA clients that will themselves be serving as CAs for other instances of InterSystems IRIS.
 - **Code signing** — For CA clients that perform code signing.
6. **Important:** This step requires that you verify the identity of the technical contact for the potential CA client using a means that prevents impersonation.

As the instructions on this page specify, you must contact the designated technical contact for the instance that has submitted the CSR. According to the policies of your organization, contact this person by phone or in person and verify:

- This person's identity
- This person's authority to hold a certificate containing the Subject Distinguished Name shown above, signed by the CA for which you are responsible
- That the SHA-1 fingerprint shown above matches the one reported to them when they submitted the certificate signing request

7. Once you have specified the purpose of the certificate and verified the relevant information with the technical contact, you can issue the certificate. To do this, click **Issue Certificate**. This causes the page to display the **Password for Certificate Authority's Private Key file** field.
8. In the **Password for Certificate Authority's Private Key file** field, enter the password to decrypt the CA server's private key file. If you created the private key and certificate with InterSystems IRIS, this is the value you entered in the **Password to Certificate Authority's Private Key file** field; if you created the private key and certificate using other tools, it is the password, if any, that you provided to those tools for this purpose.
9. Click **Finish** to create the certificate. If successful, InterSystems IRIS displays a message such as


```
Certificate number 31 issued for Certificate Signing Request
"Santiago Development Group"
```
10. Remove the media holding the CA server's certificate and private key, and store it in a secure location.

InterSystems IRIS has now created the certificate and notified the technical contact for the CA client by email that the certificate is available for download. The CA client's technical contact can now download the certificate to the client host.

Delete a Certificate Signing Request (CSR)

To delete a request, the procedure is:

1. In the Management Portal, go to the **Public Key Infrastructure** page (**System Administration > Security > Public Key Infrastructure**).
2. On the **Public Key Infrastructure** page, under **Certificate Authority Server**, select **Process pending Certificate Signing Requests**. This displays a table of CSRs that the CA has received and not processed or deleted; to the right of each CSR there are **Process** and **Delete** links.
3. To delete a CSR, click **Delete**. This displays a confirmation dialog.
4. In the confirmation dialog, click **OK**.
5. Complete these fields as required and click **Save**.

This deletes the CSR.

Certificate Authority Client Tasks

A certificate authority (CA) client has one-time setup tasks, which are:

1. [Configuring the InterSystems IRIS instance as a CA client](#). This involves providing location about the CA server to the potential CA client; it also includes providing contact information about the CA client's technical contact.
2. [Getting a copy of the CA certificate](#). This allows for verifying other certificates.

After setup tasks, the CA client tasks are:

1. [Submitting a certificate signing request \(CSR\) to the CA server](#). From the user's perspective, this involves specifying a distinguished name (DN) and other information. (This may happen repeatedly, if the instance has reason to have multiple distinct certificates.)
2. [Getting copies of various certificates](#). In addition to the CA client's own certificate, this includes any other certificates that the CA server has issued.

After performing these tasks, the CA client can then perform the operations that require use of the PKI. These are tasks in which it is known as an *end entity*, since it is at the end of a secured connection.

Note: Because these tasks are for the CA client administrators/technical contacts, this section is addressed to those individuals. This differs from the tasks in the Certificate Authority Server Tasks, which are addressed to CA server administrators.

Configure an InterSystems IRIS Instance as a Certificate Authority Client

The procedure to configure an InterSystems IRIS instance as a certificate authority (CA) client involves providing location about the CA server to the potential CA client; it also includes providing contact information about the CA client's technical contact. The steps are:

1. In the Management Portal, go to the **Public Key Infrastructure** page (**System Administration > Security > Public Key Infrastructure**).
2. On the **Public Key Infrastructure** page, under **Certificate Authority Client**, select **Configure Local Certificate Authority Client**. The fields on this page are:
 - **Certificate Authority server hostname** — Required. The fully qualified name of the machine of the CA server. (Specifically, this is a machine on which an instance of InterSystems IRIS is running, and this instance is serving as a CA server. It must be configured as a CA server prior to configuring any instance as a CA client.)
 - **Certificate Authority WebServer port number** — Required. The webserver port number for the instance of InterSystems IRIS serving as the CA server.
 - **Certificate Authority server path** — Required. The path of the web service of the CA server. By default, this is `/isc/pki/PKI.CAServer.cls`. (This value is used along with the server hostname and port number to contact the web service for the CA.)
 - **Local technical contact** — The person who provides verification information to the CA server on behalf of the CA client. For this person, the following information is required:
 - **Name** — Required. The name of the technical contact for the CA client.
 - **Phone number** — The contact's phone number. This is so that the CA administrator can contact the CA client's technical contact to perform verification prior to issuing the CA client's certificate. The phone number is not required, since InterSystems IRIS does not require a particular mechanism of verification; for example, it could happen in person.
 - **Email address** — The contact's email address. This is so that the CA client's technical contact can receive email notification that the CA server has processed the client's CSR and issued a certificate. The email address is not required, since the server administrator can use some other means to contact the client's technical contact about the newly issued certificate.
3. Complete these fields as required and click **Save**.

InterSystems IRIS acknowledges success through a message such as "Certificate Authority client successfully configured." At this point, the next task is to [download the CA server's certificate](#).

Submit a Certificate Signing Request to a Certificate Authority Server

Once an instance of InterSystems IRIS is configured as a certificate authority (CA) client, you can then submit a certificate signing request (CSR) to the CA server. On the surface, this involves specifying a distinguished name (DN) and other information. Under the covers, the CA client performs several actions:

1. Generating a public/private key pair.
2. Creating a Certificate Signing Request (CSR) containing the public key and a specified DN.
3. Submitting that to the CA server using a web service.

The PKI infrastructure automatically provides the CSR to the CA server, acknowledges the submission, and sends email notification to the CA server's administrator. The submission includes your contact information as the local technical contact for the CA client. The CA administrator then processes the CSR by using communications that prevent impersonation to verify the identity of the requester, the authority of the technical contact to hold a certificate with the requested DN, and the purpose for which the certificate is being issued. If the request is approved, the completion of the process includes the CA server creating a certificate.

To submit a CSR to a CA server, the procedure is:

1. In the Management Portal, go to the **Public Key Infrastructure** page (**System Administration > Security > Public Key Infrastructure**).
2. On the **Public Key Infrastructure** page, under **Certificate Authority Client**, select **Submit Certificate Signing Request to Certificate Authority Server**. The fields on this page are:
 - **File name root for local Certificate and Private Key files (without extension)** — Required. Specifies the part of the name of the private key and certificate files that is common to each. Hence, if you select `CAClient` as the file name root, the private key is stored in the `CAClient.key` file and the certificate is stored in the `CAClient.cer` file. Valid characters for this field are alphanumeric characters, the hyphen, and the underscore. The root cannot be the string "cache".
 - **Password to Certificate Authority's Private Key file and Confirm Password** — Optional. The password that for encrypting and decrypting the CA client's private key.
 - **Subject Distinguished Name** — The set of one or more name-value pairs that define the distinguished name (DN) that describes the bearer of the client certificate. You must provide a value for at least one attribute. The attributes are:
 - **Country** — A two-letter country code for the country, using the [ISO country codes](#).
 - **State or Province** — The name of the state or province, spelled out in full.
 - **Locality** — The name of the municipality, spelled out in full.
 - **Organization** — Name of the organization with which the certificate is associated. By convention, this value is, spelled out in full, such as "InterSystems Corporation," rather than simply "InterSystems" or "InterSystems Corp."
 - **Organizational Unit** — Any other organizational information, such as a department.
 - **Common Name** — A descriptive name for the client, such as "Documentation Test Client."
3. Complete these fields as required and click **Save**. If successful, InterSystems IRIS then displays a message such as:

```
SHA-1 Fingerprint:
0C:DA:5F:06:04:C7:AE:64:61:9C:5C:29:35:49:88:0D:B6:E5:7D:98,
Certificate Signing Request "CAClient060412"
successfully submitted to the Certificate Authority at instance MyCA
on node CATESTCLIENT.CATESTDOMAIN.COM
```

If InterSystems IRIS has successfully created a CSR, it has performed the following actions:

- Creating a key pair.
- Saving the private key to a file in the manager's directory with the specified file name root.
- Creating a CSR that includes the public key and saving it to a file in the manager's directory with the specified file name root.
- Submitting that CSR to the CA using the host name, port, and path specified as part of the CA client configuration process.

(If the process does not succeed, InterSystems IRIS displays an error message.)

Once the files have been created, it is strongly recommended that you store this sensitive information on encrypted, removable media that can be physically secured.

4. Make a copy of the SHA-1 fingerprint that InterSystems IRIS displays.

Important: Do not lose this information, as you will need to provide this later, as part of the verification process.

5. At this point, you have used InterSystems IRIS to create and submit the CSR. When the administrator of the CA contacts you, provide the SHA-1 fingerprint that you copied in the last step. The administrator will then create certificate for you, which you can obtain as described in [Get Certificate\(s\) from Certificate Authority Server](#).

Get Certificate(s) from Certificate Authority Server

Once a certificate authority (CA) client has been configured, it can then download any certificate associated with the CA server. This includes:

- The CA server certificate.
- Its own certificate. This is available if the CA client has submitted a certificate signing request (CSR) to the CA server and the CA server has approved the request.
- Any other certificates that the CA server has created for any other CA clients.

The procedure to obtain certificates is:

1. In the Management Portal, go to the **Public Key Infrastructure** page (**System Administration > Security > Public Key Infrastructure**).
2. On the **Public Key Infrastructure** page, under **Certificate Authority Client**, select **Get Certificate(s) from Certificate Authority server**. This displays a list of available certificates to download, as well as a button that displays certificates issued for the current instance (whether downloaded or not). Ordinarily, you need both the CA server certificate as well as your own. There are several tasks you can perform from this page:
 - To download the CA certificate, click **Get Certificate Authority Certificate**. A confirmation message such as
3. When you click **Get** to download a certificate, InterSystems IRIS displays a confirmation message, such as

```
Certificate Authority Certificate (SHA-1 Fingerprint:
E2:FB:30:09:53:90:9A:31:30:C3:F0:07:8F:64:65:CD:11:0A:1A:A2)
saved in file "c:\intersystems\myinstnace\mgr\MyCA.cer"
```

- To download any certificate that the CA has issued — including any certificate for the CA client itself, you can locate the certificate by its serial number, the name of the host of the CA client (the **Hostname** column), the name of the instance of the CA client (the **Instance** column), or the root file name of the certificate (the **Filename** column).
- To view any certificates issued for the current instance, click **Show Certificates for This Instance**. This displays a table of certificates from which you can download a certificate, listing only the **Serial Number** and **Filename** columns.

```
Certificate number 74 (SHA-1 Fingerprint:
45:E8:DE:0C:15:BF:A7:89:58:04:5E:68:2E:4D:BB:01:F5:90:94:97)
saved in file "c:\intersystems\myinstance\mgr\IstanbulAcctsPayable.cer"
```

While InterSystems IRIS initially downloads certificates to the manager's directory, once they are on the client host, you can move them anywhere.

Demo: Database Encryption

InterSystems IRIS Demo: Database Encryption

This article introduces you to how InterSystems IRIS® data platform handles database encryption, which is an important part of any organization's security strategy.

This article presents an introduction to database encryption and walks you through some initial tasks associated with creating an encrypted database. Once you've completed this guide, you will have created a key file, activated the key file, and then used it to encrypt a database. These activities are designed to use only the default settings and features, so that you can acquaint yourself with the fundamentals of the feature without having to deal with details that are off the topic (though these may be important when performing an implementation). For the full documentation on database encryption, see [Encryption Guide](#).

Why Database Encryption Is Important

While encryption does not prevent all improper or unauthorized use or disclosure of confidential or personal information, ensuring encryption of data at rest provides an important layer in the defense of the security of information. Putting encryption in place at the database level provides an added dimension to your information protection controls.

Additionally, many laws and regulations regarding sensitive or personal information recommend or require that the organization processing the data employ encryption as a first line of defense. These include laws and regulations such as:

- Health Insurance Portability and Accountability Act (HIPAA) — Requirement that Secure Protected Health information be unreadable, undecipherable, and unrecoverable
- Massachusetts 201 Code of Massachusetts Regulations (CMR) 17.00 — Requirement that personal information be encrypted in transit and at rest
- New York 23 New York Codes, Rules and Regulations (NYCRR) Part 500 — Financial and other covered organizations processing nonpublic information must utilize encryption as one means of safeguarding data
- European Union General Data Protection Regulation (GDPR) — Requirement for security safeguards to take into account encryption as a protecting control
- Italian Personal Data Protection Code (PDPC) — Section 24 of the Technical Specifications on Minimum Data Security Measures requires the processing of data disclosing health and sex life to be encrypted
- Australian Privacy Principles (APP) Principle 4 — Robust encryption implementation addresses necessary privacy enhancing technologies to secure personal information
- Japan Ministry of Economy, Trade, and Industry (METI) Guidelines — Regulatory investigation must be undertaken if compromise of personal or confidential information that was not encrypted, because under the Act on the Protection of Personal Information (APPI), Art. 20, the processor of personal information obligated to prevent leakage, loss, or damage of information

Note that many of these legal requirements focus on data breaches, as they are an increasingly common phenomenon, but the current framework obligates organizations to address risk through proper security controls, such as role-based access, password protections, intrusion detection, data loss prevention, and logging/auditing — as well as encryption. Encryption alone will not address all mandatory requirements, but provides a secure foundation. Encryption at the database level enhances protections by requiring an attacker to not only gain access to the system or file space, but to also have access to the database. This additional layer provides assurances to the organization, its customers, and any stakeholders.

How InterSystems IRIS Uses Database Encryption

For activities associated with database operations, the InterSystems IRIS encryption and decryption processes are transparent to users. From the perspective of the end user or the application developer, the application simply performs its usual

activities and the data is automatically encrypted on disk. From the perspective of the system administrator, there are a few simple tasks to ensure that data encryption occurs; after performing these tasks, again, activities occur invisibly.

What's more, these activities use a minimum of processor time, so they are likely to have no visible impact on your applications. Further, because of how our databases are constructed, these activities are highly optimized.

Encryption and decryption use the United States Government Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode, often simply known as AES CBC. InterSystems IRIS supports all legal key sizes for AES CBC: 128-, 192-, and 256-bit keys.

InterSystems IRIS performs encryption and decryption using the fastest available implementation. Whenever available, encryption and decryption take advantage of the processor-based instruction sets and their inherent efficiencies. Modern Intel and IBM POWER8 processors have such instructions. InterSystems IRIS automatically detects and uses these instructions, so you don't have take any actions to make it happen. On Intel processors, these are the Advanced Encryption Standard New Instructions (AES-NI); on IBM, they are the AES VMX instruction set.

You can store database encryption keys either on key management servers that support the key management interoperability protocol (KMIP) or in files that contain encrypted copies of the database keys. Each has its own advantages:

- KMIP is an OASIS standard protocol for clients to communicate with key management systems. KMIP servers can be specialized hardware appliances or general-purpose servers running key management software.
- When database encryption keys are stored in files, InterSystems encrypts the keys using multiple layers of the AES key wrap algorithm, with individual administrator key-encryption keys derived using the PBKDF2 algorithm, thereby making dictionary and brute force attacks impractical.

It is important to keep in mind that, while database encryption is an integral part of a security strategy, it cannot address security vulnerabilities alone. Other tools, such as protection for data in motion, are also crucial. This is why database encryption is part of the suite of tools that InterSystems IRIS provides for protecting data. These include:

- Support for government standards — You can configure InterSystems IRIS to use libraries that are validated to conform to FIPS 140-2 Federal Information Processing Standards) for database encryption. This is available on Red Hat Linux.
- Protecting selected data elements — Known as data-element encryption, this feature provides a programmatic approach that allows you to encrypt only selected parts of records, such credit card or Social Security numbers
- Protecting data in motion — InterSystems IRIS protects data in motion using the newest version of Transport-Layer Security (TLS). TLS is the industry-standard protocol for protecting data communications.
- Support for third-party authorization — InterSystems IRIS supports authorization for using resources on third-party sites, as is frequently seen on the web with logins through Facebook or Google to use a site. This is through the Open Authorization Framework version 2.0 (known as OAuth 2.0) and may include authentication through another layer, known as OpenID Connect.

Trying Database Encryption for Yourself

It's easy to use InterSystems IRIS database encryption. This simple procedure walks you through the basic steps of setting up an encrypted database.

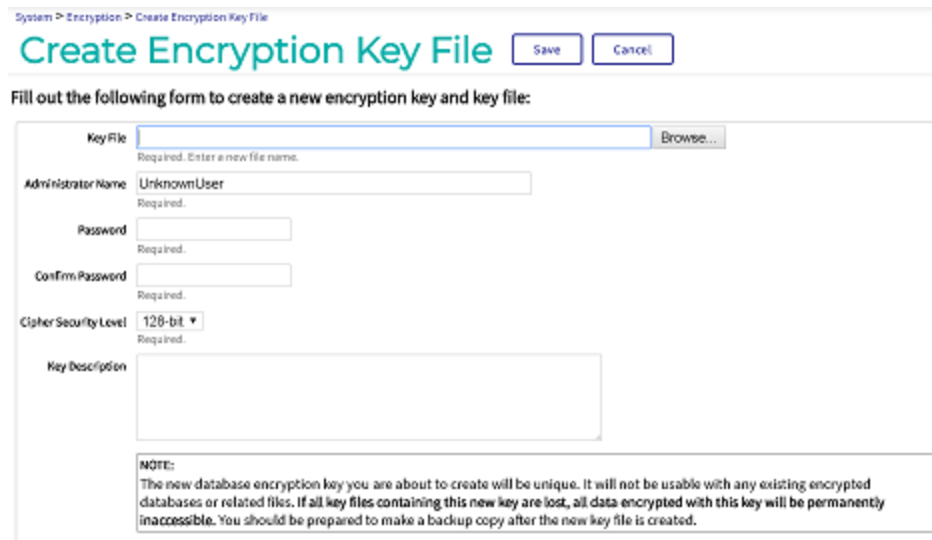
Before You Begin

To use the procedure, you will need a running instance of InterSystems IRIS. Your choices include several types of licensed and free evaluation instances. For information on how to deploy each type of instance if you do not already have one to work with, see [Deploying InterSystems IRIS](#) in *InterSystems IRIS Basics: Connecting an IDE*.

Creating an Encryption Key

First, create a key file, which automatically has a database encryption key in it:

1. Open the Management Portal for your instance in your browser, using the [URL described for your instance](#) in *InterSystems IRIS Basics: Connecting an IDE*.
2. Navigate to the **Create Encryption Key File** page (**System Administration > Encryption > Create New Encryption Key File**):



3. On the **Create Encryption Key File** page:
 - a. In the **Key File** field, enter a name and path for the key file. When you click the **Browse** button, the File Selection Dialog opens by default in the instance's `install-dir/mgr` directory (where `install-dir` is the InterSystems IRIS installation directory), for example, `C:\InterSystems\IRIS\mgr\testkeys.key`; you can use this directory on all types of instance, or select another location in the host or container file system.
 - b. In the **Administrator Name**, **Password**, and **Confirm Password** fields, enter values such as `testadmin` and `password`. This is just an example case, so don't reuse a password that you would use in a development environment.
 - c. Select the **Save** button near the top of the page.

You just created the `testkeys.key` key file in the `C:\InterSystems\` directory with a key in it that you can use for database encryption. InterSystems IRIS displays a message with the key in it, such as

New encryption key ID: 46D153E1-F895-42F9-9706-D1236115E45A

For more details about creating a key file and its initial key, see [Create a Key File](#).

Activating an Encryption Key

Next, activate the key that you just created:

1. In the Management Portal, go to the **Database Encryption** page (**System Administration > Encryption > Database Encryption**).
2. On the Database Encryption page, select the **Activate Key** button:

System > Encryption > Database Encryption

Database Encryption

[Activate Key](#) [Configure Startup Settings](#)

No database encryption key is activated.

Fill out the following form to activate a database encryption key

Key File [Browse...](#)
Required.

Administrator Name
Required.

Password
Required.

[Activate](#) [Cancel](#)

3. In the **Key File** field, enter the location of the key file you created, such as C:\InterSystems\IRIS\mgr\testkeys.key.
4. In the **Administration Name** and **Password** fields, enter the values you specified (testadmin and password).
5. Select the **Activate** button.

You can see the key ID on this page:

System > Encryption > Database Encryption

Database Encryption

[Activate Key](#) [Configure Startup Settings](#)

Activated database encryption key:

Count	Identifier	Bit length		
1	460153E1-F856-42F3-9709-D1236115E45A	128	(Default key for new encrypted databases) (Key for encrypted journal files)	Set Default Set Journal Deactivate

For more details about activating a key, see [Activate a Database Encryption Key from a Key File](#).

Creating an Encrypted Database

Now, you can create an encrypted database:

1. Again in the Management Portal, go to the **Namespaces** page (**System Administration > Configuration > System Configuration > Namespaces**).
2. On the **Namespaces** page, select **Create New Namespace**. This displays the **New Namespace** page:

System > Configuration > Namespaces > New Namespace

New Namespace

[Save](#) [Cancel](#)

Use the form below to create a new namespace:

Name of the namespace
Required.

Copy from

The default database for Globals in this namespace is a ☒ Local Database ☐ Remote Database

Select an existing database for Globals [Create New Database...](#)
Required.

The default database for Routines in this namespace is a ☒ Local Database ☐ Remote Database

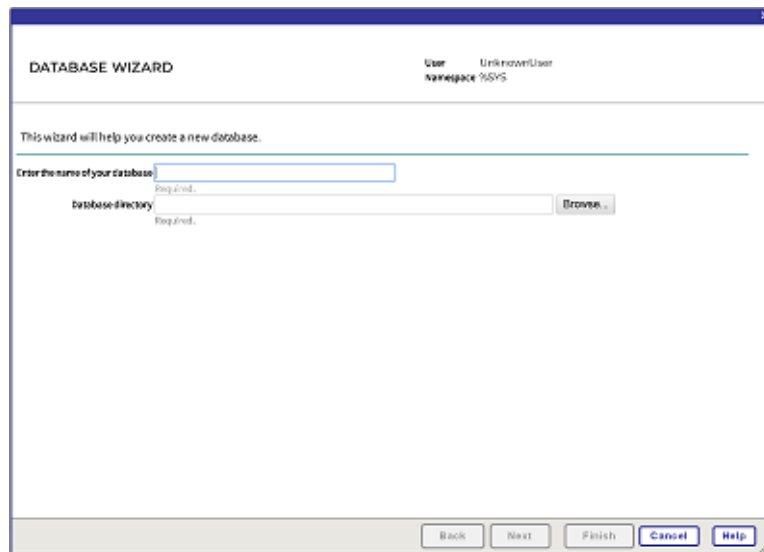
Select an existing database for Routines [Create New Database...](#)

Create a default Web application for this namespace ☒

Copy namespace mappings from

Enable namespace for interoperability productions ☒

3. On the **New Namespace** page, enter the name of the encrypted database that you are going to create, such as **ENCDB**.
4. Next to the **Select an existing database for Globals** drop-down menu, select the **Create New Database** button. This displays the **Database Wizard**:



5. On the first page of the **Database Wizard**, in the **Enter the name of your database** field, enter the name of the database you are creating, such as **ENCDB**. Enter a directory for the database, such as `C:\InterSystems\IRIS\mgr\ENCDB`. On that page, select **Next**.
6. On the next page, change the value of **Encrypt database** from **No** to **Yes**. On that page, select **Finish**.
7. Back on the **New Namespace** page, in the **Select an existing database for Routines** drop-down menu, select the database you just created, such as **ENCDB**.
8. Select the **Save** button near the top of the page and then select **Close** at the end of the resulting log.

You have now created an encrypted database called **ENCDB** that uses the key that InterSystems IRIS created when you created the key file. You can use this database just as you would use an unencrypted database. Because InterSystems IRIS hides all the machinery for encryption and decryption, you can perform all operations in the usual way and all your data will be encrypted.

For more details about creating a namespace and its associated database, see “[Create/Modify a Namespace](#)” in the “Configuring InterSystems IRIS” chapter of the *InterSystems IRIS System Administration Guide*. For background information, see “[Namespaces and Databases](#)” in the *Orientation Guide for Server-Side Programming*.

Looking at Encrypted Data

Once you have created the encrypted database, you can use it just as you would use any other, unencrypted database. The only difference is how the data is stored. To see the difference between data stored in encrypted and unencrypted databases, you can perform the following, simple test:

1. Open the Terminal for your InterSystems IRIS instance, using the [procedure described for your instance](#) in *InterSystems IRIS Basics: Connecting an IDE*.
2. Switch to the namespace for the encrypted database using the following command:

```
%SYS>set $namespace="ENCDB"
ENCDB>
```

3. In the **ENCDB** namespace, run the following command:

```
ENCDB>for i=1:1:1000 set ^x(i)="This is test number "_i
```

This creates a thousand persistent variables with content such as `This is test number 22`.

4. To confirm that you have succeeded, look at the value of one variable:

```
ENCDB>w ^x(22)
This is test number 22
ENCDB>
```

5. To open the database file, go to the directory where you created it in the previous section, such as C:\InterSystems\IRIS\mgr\ENCDB, and open the database file, IRIS.DAT. You will see content such as:

[illegible]

6. Try searching for the string “This is test number” in the file. You won’t find it — because the database is encrypted. In fact, the only unencrypted strings you’ll find are the name of the database or the identifier of its encryption key.
7. If you perform the same test on an unencrypted database, the resulting file will include content such as:

```
Cox|{[86~\Lyn?r.Ah2I2lBk/r\llYt?#2nr7abn]Uy?vdaIlloUD+g46S) iü MüDkIU0äqxeöy/iIA17mP\19iqkg\NëOBr|E'è
IG1yIqcw=wa4lZ(IF,6eA)z}cSEF#+däA4I!^~I-Jv(IÜ||A44Ct~n~Xaq21Bµ'ë~LÖI -ë(7écé(+tUIETDqI,V^ëaBaII
8CIYl!a~aëS6@MISQ~çUA$S[fäc
n0üj|az?73ya'y[AI]{-IÄRäAgG.IniuiaöAI!kIWII    »EH»,   »ags?nÇAGµCNIäâç'a+'bSnC-IIA4I'xIU!ëxâx!7C
4/6aöBçc«pñAnIU89e!*~k*4b«-ä]æe.G,}fæt I^2báiIÖ`äJLRç&VI0I|EÖ /jt çI      CySa»(IVÜUAtx<|löç|I|
7RIUIURJRD`qudII!«MOI!!«e3ä«q8     NolaUFeINU|çI«?»
ö1ë-hê|hOI{tääAyEë Npxya#7I+fv3*Nlqi|UHUöer }ezI1kIF$X~ü,+R_Q+lIä4Y'r1EIëÖlitrcI'|cy'l}wfAYç
VMZ +4 (IIT7IEIC9WIVOMFEVEV}|SPHIV-XIQ W ID(- VIP )()47)C9760-6&6=I30(6-8-1)|XR WUPUGI
@cX . ç | This is test number 10C !! This is test number 10C !! This is test number 10C !! This
```

Note that the last line of the screen shot contains the values of the variables set in the Terminal.

Other Features Related to Database Encryption

InterSystems IRIS also has other notable database encryption features that may be important for your implementation:

- **KMIP** — InterSystems IRIS allows you to store keys on servers that are separate from those that host your instances. To communicate with such servers, InterSystems IRIS supports the key management interoperability protocol (KMIP). This allows InterSystems IRIS to benefit from the standardized approach to key management that KMIP provides.
- **Changing keys and adding or removing encryption** — You can easily change a database's encryption key. It is also straightforward to encrypt an unencrypted database or make an unencrypted copy of an encrypted database, should either of these actions be necessary.
- **Encryption for related data on disk** — InterSystems allows you to easily encrypt its temporary cache databases and other on-disk content that it uses to keep its recent transaction records current (that is, its journal files).
- **Chip-based encryption** — Chips are available that perform encryption as part of their activities, which provides much faster speeds for operation. InterSystems IRIS supports the use of such chips. For more details on chip-based encryption, see the next section.

Learn More About Database Encryption

InterSystems has lots of resources to help you learn more about database encryption:

- [*Encryption Awareness*](#) — InterSystems Online Learning interactive course providing conceptual introduction to our encryption technology.
- [*Encryption Guide*](#) — InterSystems documentation on database encryption and related features.
- [*FIPS 140–2 Compliance for Database Encryption*](#) — InterSystems documentation on InterSystems IRIS support for the FIPS 140–2 standard.

TLS

TLS with the Superserver

Configuring the InterSystems IRIS Superserver to Use TLS

To use TLS for communications among components of InterSystems IRIS® data platform, configure the InterSystems IRIS superserver to use TLS. To do this, the procedure is:

1. From the Management Portal home page, go to the **SSL/TLS Configurations** page (**System Administration > Security > SSL/TLS Configurations**).
2. On the **SSL/TLS Configurations** page, select **Create New Configuration**, which displays the **New SSL/TLS Configuration** page.
3. On the **New SSL/TLS Configuration** page, create a TLS server configuration with a configuration name of %SuperServer (using the exact case as specified here). For details about creating a TLS configuration, see [Create or Edit a TLS Configuration](#).
4. On the **System-wide Security Parameters** page (**System Administration > Security > System Security > System-wide Security Parameters**), for the **Superserver SSL/TLS Support** field, choose **Enabled**. This specifies that the superserver supports (but does not require) TLS-secured connections.

Note: If you wish to configure the superserver to require TLS-secured connections, first specify that TLS is simply enabled.

5. Set up clients to use TLS as appropriate (see [Configuring InterSystems IRIS Telnet to Use TLS](#)).

TLS with Telnet

Configure the InterSystems IRIS Telnet Server to use TLS

You can configure InterSystems IRIS® to accept TLS-protected connections from Telnet clients. To do this, configure the InterSystems IRIS Telnet server to use TLS:

1. From the Management Portal home page, go to the **SSL/TLS Configurations** page (**System Administration > Security > SSL/TLS Configurations**).
2. On the **SSL/TLS Configurations** page, select **Create New Configuration**, which displays the **New SSL/TLS Configuration** page. On this page, create a TLS configuration with a configuration name of %TELNET/SSL.
3. Enable the Telnet service, %Service_Telnet:
 - a. On the **Services** page (**System Administration > Security > Services**), click %Service_Telnet to display the **Edit Service** page for the Telnet service.
 - b. On the **Edit Service** page, check **Service Enabled** if it is not already checked.
 - c. Click **Save**.
4. On the **System-wide Security Parameters** page (**System Administration > Security > System Security**), select **Enabled** for both the **Superserver SSL/TLS support** and the **Telnet server SSL/TLS support** settings.

Configuring Telnet Clients to Use TLS

InterSystems IRIS accepts TLS connections from both the InterSystems Telnet client and third-party Telnet clients.

Configure the InterSystems Telnet Client to Use TLS

You can configure the InterSystems Telnet client to use a TLS connection. The process involves several steps:

1. On the instance that is the Telnet server, configure it according to the [instructions in the previous section](#), which includes the option of requiring TLS.
2. On the instance that is the Telnet client, configure the settings file according to the instructions in “[Connecting from a Windows Client Using a Settings File](#).”

Configure Third-Party Telnet Clients to Use TLS

You can configure third-party Telnet clients to connect to an InterSystems Telnet server. The required or recommended configuration actions depend on the software in use and the selected cipher suites. The following guidelines apply:

- If the Telnet client requires server authentication, then the server must provide a certificate and the client must have access to the server’s certificate chain.
- If the InterSystems IRIS Telnet server *requires* client authentication, then the client must provide a certificate and the server must have access to the client’s certificate chain.
- If the InterSystems IRIS Telnet server *requests* client authentication, then the client has the option of providing a certificate and a certificate chain to its certificate authority (CA). If the client does not provide a certificate, then authentication succeeds; if it provides a non-valid certificate or certificate chain, then authentication fails.

For information on how certificate and certificate chains are used for authentication, see [Establishing the Required Certificate Chain](#).

TLS with Python Clients

Configuring Python Clients to Use TLS with InterSystems IRIS

You can configure a Python client application to use TLS when it communicates with InterSystems IRIS® data platform. To establish a Python connection using TLS:

1. Configure the superserver to use TLS as described in [Configuring the InterSystems IRIS Superserver to Use TLS](#).
2. Ensure that you have installed any relevant CA certificates for verifying the server certificate.
3. Configure the Python client based on your version. Note that version 4 uses the SSLDefs.ini file for the SSL configuration. For more information on how to configure this file, see [Connecting from a Windows Client Using a Settings File](#).

V3

```
import ssl
import iris

context = ssl.SSLContext(ssl.PROTOCOL_TLS)
context.verify_mode = ssl.CERT_REQUIRED
cafile = "path/to/CACert.pem"
context.load_verify_locations(cafile)
context.load_cert_chain("path/to/Cert.pem", "path/to/Key.pem", "apasswordifany")

connection =
    iris.createConnection("127.0.0.1", 1972, "user", "_SYSTEM", "SYS", 10000, sslcontext=context)
...
connection.close()
```

V4

```
import iris
# On Windows lookup is based on address-port pair in SSLDefs.ini. -- GDConfig will be used
connection =
    iris.createConnection("127.0.0.1", 1972, "user", "_SYSTEM", "SYS", 10000, sslconfig=True)

# On Unix lookup is based on a provided configuration name instead of address-port pair. -- GDConfig2
# will be used
# connection =
#     iris.createConnection("127.0.0.1", 1972, "user", "_SYSTEM", "SYS", 10000, sslconfig="GDConfig2")
...
connection.close()
```

Below is an example of the SSLDefs.ini file for a Python client configuration as used in the V4 code above:

SSLDefs.ini

```
[IRIS]
Address=127.0.0.1
Port=1972
SSLConfig=GDConfig

[GDConfig]
TLSMinVersion=16
TLSMaxVersion=32
KeyType=2
VerifyPeer=0
CipherList=ALL:!aNULL:!eNULL:!EXP:!SSLv2
Ciphersuites=TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
Password=apasswordifany
CertFile=path/to/Cert.pem
KeyFile=path/to/Key.pem
CAfile=path/to/CACert.pem

[GDConfig2]
TLSMinVersion=16
TLSMaxVersion=32
KeyType=2
VerifyPeer=0
CipherList=ALL:!aNULL:!eNULL:!EXP:!SSLv2
Ciphersuites=TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
Password=apasswordifany
```



```
CertFile=path/to/AnotherCert.pem  
KeyFile=path/to/AnotherKey.pem  
CAfile=path/to/AnotherCACert.pem
```

Note: As of this writing, you cannot change the Python TLSv1.3 ciphers. For TLSv1.3, only Ciphersuites is used and its value must be exactly this list (order may vary):

```
TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256
```

For versions earlier than TLSv1.3, only CipherList is used.

TLS with Java Clients

Configuring Java Clients to Use TLS with InterSystems IRIS

You can configure a Java client application to use TLS when it communicates with InterSystems IRIS® data platform. This communication occurs through the superserver, so a related required step is setting up the superserver to use TLS; this is described in [Configuring the InterSystems IRIS Superserver to Use TLS](#). Java clients can be implemented using either JDBC or object bindings.

The process for configuring a Java client application to use TLS with InterSystems IRIS is:

1. Determine if the client requires a keystore or a truststore. This depends on several factors: whether or not the InterSystems IRIS server requests or requires client authentication; whether server authentication is required; and the cipher suites in use. See “[Determine the Need for a Keystore and a Truststore](#)” for more information.
2. Create a configuration file with properties in it to provide those features. See “[Create a Client Configuration](#)” for more information.
3. In the code for the client application, optionally specify the name of the client configuration; if you do not specify a name, Java uses the default configuration information. See “[Specify the Use of the Client Configuration](#)” for more information.

Determine the Need for a Keystore and a Truststore

A keystore serves as a repository for the client’s private key, public key certificate, and any Certificate Authority (CA) information. This information is needed (1) if the InterSystems IRIS server requires client authentication or (2) if the cipher suite in use requires a client key pair:

- Whether or not the InterSystems IRIS server requires client authentication is determined by the choice for the **Peer certificate verification level** field on the **Edit SSL/TLS Configuration** page for that InterSystems IRIS instance’s “%SuperServer” TLS configuration. If the field has a value of **Require**, the client must have a certificate; if the field has a value of **Request**, the server checks a certificate if one is available.
- The client and server agree upon a cipher suite to use. This cipher suite determines whether or not there is a client certificate, a key pair, or both. The enabled server cipher suites are determined by the value of the **Enabled ciphersuites** field on the **Edit SSL/TLS Configuration** page for that InterSystems IRIS instance’s “%SuperServer” TLS configuration. The cipher suites available to the client depend on the version of Java it is using.

If the client has a private key and certificate, these are stored in the client’s keystore; the keystore can also hold the client’s root CA certificate and any intermediate CA certificates. To authenticate the server, the client may need to have the root CA certificate for the server and any intermediate CA certificates, these can be stored either in the client’s truststore or along with client certificate information in the keystore. For more information on keystores and truststores, see the section “Keystores and Truststores” in the [Java Secure Socket Extension \(JSSE\) Reference Guide](#).

Create a Client Configuration

The behavior of a Java client depends on the values of properties in its configuration. The configuration gets these values from what is known as a “configuration file,” either from the configuration file’s default values or from its configuration-specific values. The following sections describe how configuration files work:

- [Configuration Files, Configurations, Properties, Values, and Defaults](#)
- [Java Client Configuration Properties](#)
- [A Sample Configuration File](#)
- [Name the Configuration File](#)

Configuration Files, Configurations, Properties, Values, and Defaults

Each configuration file specifies values for the properties that one or more configurations use. The file includes both default values and configuration-specific values, in the form of name-value pairs. Generally, unversioned property names specify default values for properties and versioned property names specify configuration-specific values.

If a configuration file contains only one configuration definition, that single configuration can use unversioned properties. However, it cannot have an associated name property. Without a named configuration, invoke the configuration without specifying a name (as described in “[Specify the Use of the Client Configuration](#)” and “[Specify a Configuration without a Name](#)”).

If a configuration file contains multiple configurations, each configuration is defined by the existence of a numbered version of the *name* property of the form *name.n*, where *n* is the number of the configuration. The names of a configuration’s other properties use the same version number as the *name* property, so that they have a form of *propertyname.n* where *propertyname* is the name of the property and *n* is the number of the configuration.

The definitions in a configuration file are case-sensitive. Their use of spaces is flexible. The order of property definitions is also flexible.

To specify the default value of a property for use by all configurations, specify an unversioned property name and its value in the following form:

```
propertyName = propertyValue
```

For example, to specify the default value for the `keyStoreType` property as `pkcs12`, the form is:

```
keyStoreType = pkcs12
```

To override the default value for a property, specify a versioned property name, such as:

```
keyStoreType.1 = jceks
```

If a configuration file contains multiple configuration definitions, then these versions must use sequential ordering; if client application code refers to a configuration that follows a sequential gap, then an error results. For example, suppose that a configuration file has three versioned name properties: `name.1`, `name.2`, and `name.4`; the configuration associated with the `name.4` property will not ever be created and a reference to it will fail with an error.

Java Client Configuration Properties

The properties are:

- `cipherSuites` — A comma-delimited list of supported cipher suites. The available cipher suites depend on the JRE (Java Runtime Environment) on the machine. During the TLS handshake, the server selects the strongest cipher suite that both it and the client support. [Optional]
- `debug` — Whether or not debugging information is logged to the Java `system.err` file. This property can have a value of `true` or `false` (`false` is the default). The setting of this property has no effect on exception handling. [Optional]
- `keyRecoveryPassword` — Password used to access the client’s private key; this was created at the same time as the private key pair. [Required if the private key has password protections and application code is not passing in the private key as an input parameter.]
- `keyStore` — The file for storing the client private key and certificate information. The keystore can also hold the content typically associated with the truststore. [Optional]
- `keyStorePassword` — Password to gain access to the keystore. [Required if a password was specified when the keystore was created.]
- `keyStoreType` — The format of the keystore file, if one is specified. [Optional]

Supported formats are:

- `jks` — Java KeyStore, the Java proprietary format. [Default]
 - `jceks` — Java Cryptography Extension KeyStore format.
 - `pcks12` — Public Key Certificate Standard #12 format.
- `logFile` — The file in which Java records errors. [Optional]
 - `name` — A versioned identifier for the Java client configuration. (Each name property must be versioned. Any unversioned name property is not meaningful and is ignored.) [Optional]

If the configuration file specifies only a single configuration and only uses unversioned property names, the name property is not required (as described in “[Specify the Use of the Client Configuration](#)”). For information about specifying multiple configurations with a single configuration file, see [Configuration Files, Configurations, Properties, Values, and Defaults](#)).

- `protocol` — The version of the TLS protocol to be used for the connection. [Required]

Supported values include:

- `TLS` — Any version of the TLS protocol. During the TLS handshake, the server selects the latest (most recent) version of the protocol that it supports. [Default]
 - `TLSv1` — Version 1 of TLS.
 - `TLSv1.1` — Version 1.1 of TLS.
 - `TLSv1.2` — Version 1.2 of TLS.
 - `TLSv1.3` — Version 1.3 of TLS.
- `serverHostNameVerification` — Whether or not the connection performs server hostname verification to prevent man-in-the-middle attacks. This property can have a value of `true` or `false` (`false` is the default). [Optional]
 - `trustStore` — The file for storing the server’s root CA certificate; it can also hold the certificates for any intermediate CAs. (This information can also be placed in the keystore.) [Optional]
 - `trustStorePassword` — Password to gain access to the truststore. [Required if a password was specified when the keystore was created.]
 - `trustStoreType` — The format of the truststore file, if one is specified. [Optional]

Supported formats are:

- `jks` — Java KeyStore, the Java proprietary format. [Default]
- `jceks` — Java Cryptography Extension KeyStore format.
- `pcks12` — Public Key Certificate Standard #12 format.

A Sample Configuration File

The following is a sample configuration file for use with a Java client:

```
debug = false
logFile = javatls.log
protocol = TLSv1.3
cipherSuites = TLS_AES_256_GCM_SHA384
keyStoreType = JKS
keyStore = keystore.jks
keyRecoveryPassword = <password>
keyStorePassword = <password>
trustStoreType = JKS
trustStore = truststore.jks
trustStorePassword = <password>
trustStoreRecoveryPassword = <password>
```

```

name.1 = IRISJavaClient1
keyStorePassword.1 = <password>
keyRecoveryPassword.1 = <password>
trustStorePassword.1 = <password>
trustStoreRecoveryPassword.1 = <password>

name.2 = IRISJavaClient2
protocol.2 = TLS
keyStoreType.2 = pkcs12
keyStore.2 = keystore.p12
keyStorePassword.2 = <password>
trustStore.2 = cjcl.ts
trustStorePassword.2 = <password>

name.3 = IRISJavaClient3
protocol.3 = TLSv1.2
debug.3 = true
cipherSuites.3 = TLS_RSA_WITH_AES_128_CBC_SHA

```

Name the Configuration File

Either save the configuration file with the name `SSLConfig.properties` or set the value of the Java environment variable `com.intersystems.SSLConfigFile` to the name of the file. The code checks for the file in the current working directory.

Specify the Use of the Client Configuration

Once a configuration has been defined, client application code invokes it when connecting to the server. You can do this either with calls for the [DriverManager](#) object or the [IRISDataSource](#) object.

Use the DriverManager Object

With `DriverManager`, this involves the following steps:

1. Creating a Java Properties object.
2. Setting the value for various properties of that object.
3. Passing that object to Java Connection object for the connection from the client to the InterSystems IRIS server.

To specify information for the connection, the first part of the process is to create a Properties object from a configuration file and set the values of particular properties in it. In its simplest form, the code to do this is:

```

java.util.Properties prop = new java.util.Properties();
prop.put("connection security level", "10");
prop.put("SSL configuration name", configName);
prop.put("key recovery password", keyPassword);

```

where

- The connection security level of 10 specifies that the client is attempting use TLS to protect the connection.
- `configName` is a variable whose value holds the name of Java client configuration. If the configuration file has only default values and these are being used for a single configuration, do not include this line; see [Specify a Configuration without a Name](#) for details.
- `keyPassword` is the password required to extract the client's private key from the keystore.

Once the Properties object exists and has been populated, the final step is to pass it to the connection from the InterSystems IRIS Java client to the InterSystems IRIS server. This is done in the call to the **`DriverManager.getConnection`** method. The form of this call is:

```

Connection conn = DriverManager.getConnection(IRISServerAddress, prop);

```

where `IRISServerAddress` is a string that specifies the address of the InterSystems IRIS server and `prop` is the properties object being passed to that string.

If this call succeeds, the TLS-protected connection has been established. Typically, application code containing calls such as those described in this section includes various checks for success and protection against any errors. For more details about using InterSystems IRIS Java connectivity, see *Using Java JDBC with InterSystems IRIS*.

Use the IRISDataSourceObject

With the IRISDataSource object, the procedure is to create the object, call its methods to set the relevant values, and establish the connection. The methods are:

- **setConnectionSecurityLevel** — This method takes a single argument: a connection security level of 10, which specifies that the client is attempting use TLS to protect the connection.
- **setSSLConfigurationName** — This method takes a single argument: a variable whose value holds the name of Java client configuration. If the configuration file has only default values and these are being used for a single configuration, do not include this line; see [Specify a Configuration without a Name](#) for details.
- **setKeyRecoveryPassword** — This method takes a single argument: the password required to extract the client's private key from the keystore.

In its simplest form, the code to do this is:

```
try{
    IRISDataSource ds = new IRISDataSource();

    ds.setURL("jdbc:IRIS://127.0.0.1:1972/TESTNAMESPACE");
    ds.setConnectionSecurityLevel(10);
    ds.setSSLConfigurationName(configName);
    ds.setKeyRecoveryPassword(keyPassword);

    Connection dbconnection = ds.getConnection();
}
```

For a complete list of the methods for getting and setting properties, see the JDBC Quick Reference. The JavaDoc for `com.intersystems.jdbc.IRISDataSource` is under `<install-dir>/dev/java/doc/index.html`

Specify a Configuration without a Name

If a configuration file contains only one configuration definition, that single configuration can use unversioned properties. However, it cannot have an associated name property.

When working with a DriverManager object, the Properties object uses only the default values from the configuration file. The code for creating this object differs from the typical case in that there is no call to specify a value for the “SSL configuration name” key:

```
java.util.Properties prop = new java.util.Properties();
prop.put("connection security level", "10");
prop.put("key recovery password",keyPassword);
```

When working with an IRISDataSource object, if you want to specify an unnamed configuration, simply do not call **setSSLConfigurationName**.

TLS with .NET Clients

Configuring .NET Clients to Use TLS with InterSystems IRIS

InterSystems IRIS® data platform supports TLS connections from .NET clients.

To establish a .NET connection that uses TLS:

1. If you have not done so already, [configure the InterSystems IRIS superserver to use TLS](#) so it can accept TLS connections from the .NET client.
2. [Create a TLS configuration](#) for the .NET client.
3. Ensure that you have installed any relevant CA certificates for verifying the server certificate. The location for these is the current user's certificate store (Certificates – Current User\Trusted Root Certification Authorities).
4. Establish a connection to a server, based on the format of the connection string as described in the Creating a Connection section of “Connecting to the InterSystems Database”. In addition to the name-value pairs for the server, port, and namespace, include the *SSL* keyword and specify its value as `true`. For example, a connection that uses TLS protection might have a connection string of the form:

```
IrisConnect.ConnectionString =  
    "Server=localhost; Port=1972; Namespace=TESTNAMESPACE; SSL=true;"  
    + "Password=SYS; User ID=_SYSTEM;"
```

The `true` value of the *SSL* keyword specifies that TLS secures the client-server connection (by authenticating the InterSystems IRIS server to the .NET client and, optionally, authenticating the client to the server). Once the secure connection is established, the InterSystems IRIS server uses the *User ID* and *Password* keywords to authenticate the identity of the user connecting through the .NET client. (Note that the connection string does not specify anything related to mutual authentication; it merely specifies a server, which in turn may request or require client authentication.)

TLS with Studio

Configuring Studio to Use TLS with InterSystems IRIS

You can configure Studio to use a TLS connection. The process involves several steps:

1. On the instance acting as a TLS server and accepting the connection from Studio:
 - a. Set up the %SuperServer TLS configuration. For more information on this process, see “[Configuring the InterSystems IRIS Superserver to Use TLS](#).”
 - b. Enable TLS connections for superserver clients (because Studio is a superserver client).

Specifically, on the **System-wide Security Parameters** page (**System Administration** > **Security** > **System Security** > **System-wide Security Parameters**), for the **Superserver SSL/TLS Support** field, choose **Enabled**.
2. On the Windows machine where Studio is running (which is acting as a TLS client), configure the settings file for the connection from Studio to the TLS server instance. For more information on this process, see [Connecting from a Windows Client Using a Settings File](#).

TLS and Windows with .ini File

Connecting from a Windows Client Using a Settings File

If you are on Windows and are using Studio, ODBC, or the Terminal as a TLS client, you can use a settings file to configure connections and configurations. This mechanism is available even if there is no instance of InterSystems IRIS® data platform on the host.

To use a settings file:

1. Get the certificate authority (CA) certificate for the server. Store it on disk and note the location — you will use it later.
2. Create a file containing connection definitions and configuration definitions, as described in the [About the Settings File](#) section.
3. Name the file `SSLDefs.ini` and place it in the `InterSystems\IRIS` directory in the directory for 32-bit common program files. Typically, this is the `C:\Program Files (x86)\Common Files\InterSystems\IRIS\` directory; if you need to locate the directory, check the value of the Windows environment variable `CommonProgramFiles(x86)` on 64-bit Windows or `CommonProgramFiles` on 32-bit Windows.

By creating the file and placing it in this location, it will automatically be used when you connect to a host and a port that match one of the connections listed in the file.

Note: Use of the settings file (`SSLDefs.ini`) has the following restrictions:

1. The settings file is only for connections that use the `irisconnect.dll` or `irisconnect64.dll` executable (which are for 32-bit and 64-bit machines, respectively). Connections that use other mechanisms (such as for ADO) do not use the settings file.
2. Connections from a Windows client to InterSystems IRIS that use the settings file do not support Kerberos authentication.

About the Settings File

A settings file holds specifications for both connections to TLS servers and the TLS configurations that those connections use. For each Windows host that is a TLS client, a single file holds all its connections and configurations. The necessary information to create a file is:

- [The Syntax of the Settings File](#)
- [Connection Properties](#)
- [Configuration Properties](#)

The Syntax of the Settings File

The settings file contains one or more *connection definitions* and one or more *configuration definitions*:

- Each definition begins with an identifier for the connection or configuration. This appears in brackets on its own line, such as:

```
[MyConfiguration]
```

The identifier can include spaces and punctuation, such as:

```
[MyOtherConfiguration, which connects outside of my local network]
```

- Each definition ends either with the next bracketed identifier or the end of the file.

- Each definition includes multiple key-value pairs. All of these use the syntax:

key=value

- The group of key-value pairs specify the properties of a connection definition or configuration definition.
- The value in each key-value pair appears unquoted.

Connection Definitions

Each settings file contains one or more *connection definitions*, each of which specifies the properties a TLS connection and matches that connection to a TLS configuration. The first line of a connection definition is its identifier, which appears in brackets. After the identifier, there are multiple lines specifying information about the TLS server and the connection to it:

Address

Required. The address of the TLS server. This can be an IP address, an unqualified host name in the local domain, or a fully-qualified hostname. (Note: The client only uses the specified configuration if the values of both Address and either Port or TelnetPort match the server to which the client application is connecting.)

Port

Required. The port number on which the TLS server accepts connections. (Note: The client only uses the specified configuration if the values of both Address and either Port or TelnetPort match the server to which the client application is connecting.)

TelnetPort

The port number on the TLS server that accepts TLS-protected connections for InterSystems Telnet. If you do not specify this value, connections using InterSystems Telnet do not support TLS. (Note: The client only uses the specified configuration if the values of both Address and either Port or TelnetPort match the server to which the client application is connecting.)

SSLConfig

Required. The TLS configuration that the client uses when connecting to the server specified in this definition. Each configuration is defined in its own section.

Configuration Definitions

Each settings file contains one or more *configuration definitions*, each of which specifies the properties of a TLS configuration; for more information on TLS configurations, see “[About Configurations](#).” The first line of a configuration definition is its identifier, which appears in brackets; if the configuration identifier appears as the value of a connection definition’s SSLConfig property, the connection uses the configuration to specify its behavior. After the identifier, there are multiple lines specifying the value of each of the configuration’s properties:

Protocols

Deprecated. Use TLSMinVersion and TLSMaxVersion instead.

The versions of the TLS protocol(s) that the configuration supports, where each version of the protocol has a numeric value as listed in TLSMinVersion and TLSMaxVersion. To specify support for multiple versions of the protocol, use the sum of their values. Hence, to specify support for TLS v1.1 and TLS v1.2, use a value of 24.

This property is equivalent to the **Protocols** field in the TLS configuration page in the Management Portal.

TLSTMinVersion

Required for configurations that support v1.3. The earliest version of the TLS protocol that this configuration supports, where each version of the protocol has a numeric value and supported versions are:

- TLS v1 — 4
- TLS v1.1 — 8
- TLS v1.2 — 16
- TLS v1.3 — 32

This property is equivalent to the **Minimum Protocol Version** field in the TLS configuration page in the Management Portal.

TLSTMaxVersion

Required for configurations that support v1.3. The most recent version of the TLS protocol that this configuration supports, where each version of the protocol has a numeric value and supported versions are:

- TLS v1 — 4
- TLS v1.1 — 8
- TLS v1.2 — 16
- TLS v1.3 — 32

This property is equivalent to the **Maximum Protocol Version** field in the TLS configuration page in the Management Portal.

VerifyPeer

Required. Whether or not the client requires the verification of the certificate of the server to which it is connecting:

- 0 — Does not require (and does not perform) peer verification; the connection is established under all circumstances.
- 1 — Requires peer verification; the connection is established only if verification succeeds. This is the recommended value; if you choose this value, you must specify a value for the CAFile property.

This property is equivalent to the **Server certificate verification** field in the TLS configuration page in the Management Portal.

VerifyHost

Whether or not the client checks if the Common Name or subjectAlternativeName fields of the server's certificate match the host name or IP address as specified in the connection definition:

- 0 — Does not check.
- 1 — Checks.

This property does not have an equivalent in the Management Portal. However, it is the same type of check as the SSLCheckServerIdentity property of the %Net.HttpRequest class.

CipherList

Required for configurations that support connections using TLS v1.2 or earlier. The set of cipher suites that the client supports for encryption and hashing. For information on this property's syntax, see the OpenSSL documentation on the [ciphers](#) command.

The default value is `ALL : !aNULL : !eNULL : !EXP : !SSLv2`, and InterSystems strongly suggests using this value. For more information about this syntax in InterSystems IRIS, see [Enabled Cipher Suites Syntax](#).

This property is equivalent to the **Enabled ciphersuites** field in the TLS configuration page in the Management Portal.

Ciphersuites

Required for configurations that support connections using TLS v1.3. The set of ciphers that the client supports for encryption and hashing. For information on this property's syntax, see the OpenSSL documentation on the [ciphers](#) command.

InterSystems strongly recommends using a value of

`TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256`, which is the default. For more information about this syntax in InterSystems IRIS, see [Enabled Cipher Suites Syntax](#).

CertFile

The absolute path and name of the file that contains the client's trusted certificate authority (CA) file; if the client does not have a CA, do not specify a value for this property. If specified, this is an X.509 certificate(s) in PEM format and can include a certificate chain. For information on how this value is used, see [Establishing the Required Certificate Chain](#). (Note that certificates from the Windows Certificate Export Wizard must be in PEM-encoded X.509 format, not the default of DER encoded binary X.509.)

This property is equivalent to the **File containing this client's certificate** field in the TLS configuration page in the Management Portal.

KeyFile

The absolute path and name of the configuration's private key file; if the client does not have a private key, do not specify a value for this property.

This property is equivalent to the **File containing associated private key** field in the TLS configuration page in the Management Portal.

Password

The password for decrypting the configuration's private key. If you are using a private key with a password, this property is required; if you are not using a certificate for the client or if the private key does not have a password, do not specify a value for this property. (If the private key is password-protected and you do not provide a value here, InterSystems IRIS cannot decrypt and use the private key.)

This property is equivalent to the **Private key password** field in the TLS configuration page in the Management Portal.

KeyType

If the configuration has a private key and certificate, the format in which the configuration's private key is stored:

- DSA — 1
- RSA — 2

This property is equivalent to the **Private key type** field in the TLS configuration page in the Management Portal.

CAfile

Required. The absolute path and name of the file that contains the server's trusted certificate authority (CA) file. This is an X.509 certificate(s) in PEM format. Note that:

- If you have specified a `VerifyPeer` value of 1, you must provide this value.
- This is the certificate for CA of the server to which you are connecting, *not* the certificate for your CA.

This property is equivalent to the **File containing trusted Certificate Authority certificate(s)** field in the TLS configuration page in the Management Portal. However, unlike the Portal, it does not support the use of the `%OSCertificateStore` string.

A Sample Settings File

The following sample file defines two connections and two configurations:

```
[MyServer1 TLS to an InterSystems IRIS instance with TLS-protected InterSystems Telnet]
Address=myserver1
Port=57777
TelnetPort=23
SSLConfig=TLSConfig

[MyServer2 TLS to an InterSystems IRIS instance using TLSv1.2]
Address=myserver2.myexample.com
Port=57777
SSLConfig=TLSv1.2only

[TLSConfig]
TLSMinVersion=16
TLSMaxVersion=32
CipherList=ALL:!aNULL:!eNULL:!EXP:!SSLv2
Ciphersuites=TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
KeyType=2
VerifyPeer=1
Password=
CertFile=c:\InterSystems\certificates\nopwcllicert.pem
KeyFile=c:\InterSystems\certificates\nopwclikey.pem
CAfile=c:\InterSystems\certificates\cacert.pem

[TLSv1.2only]
TLSMinVersion=16
TLSMaxVersion=16
CipherList=ALL:!aNULL:!eNULL:!EXP:!SSLv2
KeyType=2
VerifyPeer=1
Password=
CertFile=c:\InterSystems\certificates\nopwcllicert.pem
KeyFile=c:\InterSystems\certificates\nopwclikey.pem
CAfile=c:\InterSystems\certificates\cacert.pem
```

How It Works

Important: This section describes how InterSystems products use a settings file to establish a TLS connection. By describing the mechanisms in use, it includes alternate means of creating a TLS connection. InterSystems recommends that you use the standard approach described above, rather than the alternatives mentioned here.

InterSystems IRIS uses the settings file as follows:

1. When you attempt to establish a TLS connection, the InterSystems IRIS TCP/IP client connection library locates the settings file containing connection definitions and configurations. This file is `irisconnect.dll` on 32-bit machines and `irisconnect64.dll` on 64-bit machines. To do this:
 - a. It checks the Windows registry for any TLS connection definitions.
 - b. If there are no connection definitions in the registry, the library attempts to locate any TLS configurations that are stored in a settings file.
 - c. If the `ISC_SSLconfigurations` environment variable exists, the library uses the value of that variable as the full path and file name of the settings file.

Note: If you need to define the value of the *ISC_SSLconfigurations* environment variable, you may need administrator permissions.

- d. If the *ISC_SSLconfigurations* environment variable does not exist, the library uses the *SSLdefs.ini* file in the *InterSystems\IRIS* directory under the 32-bit common program files directory identified by the Windows environment variables *CommonProgramFiles(x86)* on 64-bit Windows or *CommonProgramFiles* on 32-bit Windows.

- 2. Once it has located the settings file, the library locates the relevant connection definition for the connection you are attempting to establish.

To do this, it searches the sections of the file for one that contains *Address* and *Port* properties that match those of the connection you are attempting to establish. When it locates such a section, it uses the value of the *SSLConfig* property there to locate the matching TLS configuration section.

- 3. In the specified TLS configuration section, the library uses the values of the configuration properties to specify the type of connection to initiate with the server.

Configuring InterSystems IRIS to Use TLS with Mirroring

About Mirroring and TLS

For general information about InterSystems IRIS® data platform support for mirroring, see [Mirroring](#).

To provide security within a mirror, you can configure its nodes to use TLS. This provides for both authentication of one node to another, and for encrypted communication between nodes. As sensitive data passes between the failover members (and to an async member), it is recommended to encrypt the communication to prevent data theft or alteration over the network. Additionally, since a failover member has the ability to request an ISCAgent to take action on another InterSystems IRIS system (such as to request journal file information or force InterSystems IRIS down), it is important to protect such communication between the failover members of a mirror (and their corresponding ISCAgent processes).

Note: If the failover members use database (or journal) encryption, then TLS is required for communications between them and with any async members. (Specifically, InterSystems IRIS checks if either member has an encryption key activated; if so, the instance requires that the user enable TLS with mirroring.) For more details on database encryption and journal file encryption, see [Encryption Guide](#).

To both participate in mirroring (either as a failover member or as an async member) and use TLS, an instance must have two InterSystems IRIS TLS configurations – one of type server and the other of type client; each of these must have an X.509 TLS certificate issued by a trusted Certificate Authority. InterSystems recommends that each mirror host has its own set of certificates and that the certificates contain a unique identifier in the Common Name (CN) component of the certificate, such as the fully qualified domain name (FQDN) of the instance plus the member's InterSystems IRIS node name; because the CN is a field in a certificate's distinguished name (DN), establishing a unique CN ensures that the certificate's DN uniquely identifies the member. To create an instance's mirroring configurations, follow the procedure in the next section.

When TLS is enabled, the following actions occur:

1. **Server authentication:** When the client connects to the server, it requires the server to authenticate itself. This authentication verifies that the DN for the server's certificate matches the DN for a system configured in the client's mirror configuration. If there is no match, the client drops the connection.
2. **Client authentication:** When the server accepts a connection from a client, it requires the client to authenticate itself. This authentication also verifies that the DN for the client's matches the DN for a system configured in the server's mirror configuration. Again, if there is no match, the server drops the connection.
3. **Encryption:** The TLS protocol automatically uses the server's certificate to establish an encrypted channel between the client and the server, so that any data passing through this channel is encrypted and thereby secured.

InterSystems strongly recommends using TLS with a mirror.

Note on Configuring an Async Member with TLS

If a mirror uses TLS, then in addition to enabling TLS for the mirror and creating the configurations for each member (described in the following section), there are special steps that must be taken when configuring the second failover member or an async member; for more information, see [Authorize the Second Failover Member or Async \(TLS Only\)](#). Specifically, for each failover member, on the **Mirror Monitor** page, you need to enter the DN (distinguished name) in the **ID listed as DN in member's X.509 credentials** field; you can copy the value of the DN from **X.509 Distinguished Name** field of the **Join as Async** page (**System Administration > Configuration > Mirror Settings > Join as Async**) for the async member. (InterSystems IRIS populates the **X.509 Distinguished Name** field based on the information in the async member's certificate, making this field unavailable for manual editing.)

Note on Disabling TLS for a Mirror

To disable TLS for an existing mirror, disable it on the primary member.

Important: Use of TLS with mirroring is highly recommended. Disabling TLS for a mirror is strongly discouraged.

Create and Edit a TLS Configuration for a Mirror

To use TLS with a mirror, each member (failover or async) uses a pair of TLS configurations that are called %MirrorClient and %MirrorServer; the Portal allows you to [create](#) and [edit](#) these configurations.

Note: These configurations must already exist on each member when TLS is enabled for the mirror.

Create a TLS Configuration for a Mirror Member

To create the configurations, the procedure is:

1. Enable mirroring for that instance of InterSystems IRIS if it is not already enabled. To do this, use the **Edit Service** page for the %Service_Mirror service; on this page, select the **Service Enabled** check box. You can reach this page by either of two paths:
 - On the **Mirror Settings** page (**System Administration** > **Configuration** > **Mirror Settings**), select **Enable Mirror Service**.
 - On the **Services** page (**System Administration** > **Security** > **Services**), select %Service_Mirror.
2. Go to the **Create SSL/TLS Configurations for Mirror** page. You can do this either by:
 - On the **SSL/TLS Configurations** page (**System Administration** > **Security** > **SSL/TLS Configurations**) select **Create Configurations for Mirror**.
 - On the **Create Mirror** page (**System Administration** > **Configuration** > **Mirror Settings** > **Create Mirror**) select **Set up SSL/TLS**.
3. On the **Create SSL/TLS Configurations for Mirror** page, complete the fields on the form. The fields on this page are a subset of those on the **New SSL/TLS Configuration** page (as described in [Create or Edit a TLS Configuration](#)). Since this page creates both server and client configurations that mirroring automatically enables (%MirrorClient and %MirrorServer), there are no **Configuration Name**, **Description**, or **Enabled** fields; also, for the private-key password, this page allows you to enter or replace one (**Enter new password**), specify that none is to be used (**Clear password**), or leave an existing one as it is (**Leave as is**).

Since both configurations need the same X.509 credentials, completing this form saves both configurations simultaneously. The fields described in [Create or Edit a TLS Configuration](#) on this page are:

- File containing trusted Certificate Authority certificate(s)

Note: This file must include the certificate(s) that can be used to verify the X.509 certificates belonging to other mirror members. If the file includes multiple certificates, they must be in the correct order, as described in [Establishing the Required Certificate Chain](#), with the current instance's certificate first.

- This server's credentials:
 - File containing associated private key
 - Private key type
 - Private key password
 - Private key password (confirm)
- Cryptographic settings:
 - Minimum Protocol Version
 - Maximum Protocol Version
 - Enabled cipherlist (TLSv1.2 and below)
 - Enabled ciphersuites (TLSv1.3)

- Diffie Hellman Bits
- OCSP Settings:
 - OCSP Stapling

Once you complete the form, click **Save**.

For general information about configuring mirror members, see [Creating a Mirror](#).

Edit TLS Configurations for a Mirror Member

If you have already created a member's %MirrorClient and %MirrorServer configurations, you can edit them on the **Edit SSL/TLS Configurations for Mirror** page (**System Administration > Security > SSL/TLS Configurations**; click **Edit Configurations for Mirror**). This page displays the same fields as the **Create SSL/TLS Configurations for Mirror** page, as described in the previous section.

Special Considerations for Certificates for Mirror Members

When using TLS with mirroring, the %MirrorClient and %MirrorServer configurations must use the same certificate and private key. Hence, the certificate in use by both configurations must be usable as both a server and a client certificate.

There are certain certificate extensions that are specific to TLS clients or servers. Because the certificate in use with mirroring must be able to serve both uses (as both a client and a server), if any of these extensions appear in a certificate, then the extensions for client and server must both be present. For example, this is true for the Key Usage and Extended Key Usage extensions. If the Key Usage extension is present, then it must specify both of the following:

- The Digital Signature key usage (for clients)
- The Key Encipherment key usage (for servers)

Similarly, if the Extended Key Usage extension is present, then it must specify both:

- The Client Authentication key usage
- The Server Authentication key usage

If both extensions are present, then each must specify both values. Of course, it is also valid to have neither extension present.

If a certificate only specifies one value (either client or server), the TLS connection for mirroring fails with an error such as:

```
error:14094413:SSL routines:SSL3_READ_BYTES:ssl3 alert unsupported certificate
```

The way to eliminate this error depends on how you obtained your certificates:

- If you are using self-signed certificates, create new certificates (such as with the OpenSSL library) that adhere to these conditions.
- If you are using a commercial certificate authority tool (such as Microsoft Certificate Services), create new certificates that adhere to these conditions and use the tool to sign your certificate signing requests (CSRs).
- If you are purchasing certificates from a commercial certificate authority (such as VeriSign), include a request along with your CSRs that they adhere to these conditions.

TLS with TCP Devices

Configuring InterSystems IRIS to Use TLS with TCP Devices

This section describes how to use TLS with an InterSystems IRIS® data platform TCP connection. The process is:

1. Creating a TLS configuration that specifies the characteristics you want.
2. Opening a TCP connection or open a socket for accepting such connections.
3. Securing the connection using TLS. This can occur either as part of opening the connection/socket or afterwards.

How you invoke the InterSystems IRIS TLS functionality depends on whether you are using InterSystems IRIS as a client or server and whether you are creating an initially-secured TCP connection or adding TLS to an existing connection.

This section addresses the following topics:

- [Configure a Client to Use TLS with a TCP Connection](#)
- [Configure a Server to Use TLS with a TCP Socket](#)

Configure a Client to Use TLS with a TCP Connection

To establish a secure connection from a client, the choices are:

- [Open a TLS-secured TCP Connection from a Client](#)
- [Add TLS to an Existing TCP Connection](#)

Open a TLS-secured TCP Connection from a Client

In this scenario, InterSystems IRIS is part of the client and the TCP connection uses TLS from its inception. The procedure is:

1. Make sure that the configuration you wish to use is available. If it was created before InterSystems IRIS was last started, it is activated and ready for use; otherwise, you can [create a new one or edit an existing one](#).
2. [Open a TCP Connection Using TLS](#).

If InterSystems IRIS is a client, then it connects to the server via the client application. The connection uses the specified configuration to determine its TLS-related behavior.

Open a TCP Connection Using TLS

This involves opening a named connection that uses TLS and communicates with a particular machine and port number. The procedure is:

1. Specify the device that you are connecting to:

ObjectScript

```
Set MyConn = "|TCP|1000"
```

The TCP string specifies that this is a TCP device. For more information on initiating a TCP connection, see [OPEN Command for TCP Devices](#).

2. Open the connection, specifying the use of TLS with the `/TLS` parameter.

ObjectScript

```
OPEN MyConn: (SvrID:1000:/TLS="MyCfg")
```

where

- *MyConn* is the device previously specified
- *SvrID* can be a string that is a resolvable DNS name or an IP address
- *MyCfg* is a saved (and activated) TLS configuration

This call opens a TCP connection to the loopback processor (that is, the local machine) on port 1000 using TLS. It uses TLS according to the characteristics specified by the *MyCfg* configuration.

Optionally, the call can include a password for the private key file:

```
OPEN MyConn: (SvrID:1000:/TLS="MyCfg|MyPrivateKeyFilePassword")
```

Here, all the arguments are as above and *MyPrivateKeyFilePassword* is the actual password.

Important: The ability to include a password when Open a TCP Connection Using TLS is for real-time interactive use only. You should *never* store a private key password persistently without protecting it. If you need to store such a password, use the *PrivateKeyPassword* property of the *Security.SSLConfigs* class.

For more information on opening a TCP device, see [OPEN and USE Command Keywords for TCP Devices](#).

Once the connection is established, you can then use it in the same manner as any other TCP connection.

Add TLS to an Existing TCP Connection

This scenario assumes that the TCP connection has already been established. The procedure is:

1. Make sure that the configuration you wish to use is available. If it was created before InterSystems IRIS was last started, it is activated and ready for use; otherwise, you can [create a new one or edit an existing one](#).
2. [Securing the existing TCP connection using TLS](#).

Secure an Existing TCP Connection Using TLS

This involves adding TLS to an already-existing connection to a particular machine and port number. The procedure is:

1. Determine the name of the device to which there is a connection. For example, this might have been established using the following code:

```
SET MyConn="|TCP|1000"
OPEN MyConn: ("localhost":1000)
```

The TCP string specifies that this is a TCP device. For more information on initiating a TCP connection, see [OPEN Command for TCP Devices](#).

2. Specify the use of TLS as follows with the */TLS* parameter:

```
USE MyConn: (:/TLS="MyCfg")
```

where

- *MyConn* is the device previously specified
- *MyCfg* is a TLS configuration

Optionally, the call can include a password for the private key file:

```
USE MyConn: (:/TLS="MyCfg|MyPrivateKeyFilePassword")
```

Here, all the arguments are as above and *MyPrivateKeyFilePassword* is the actual password.

Important: The ability to include a password when securing an existing TCP connection using TLS is for real-time interactive use only. You should *never* store a private key password persistently without protecting it. If you need to store such a password, use the PrivateKeyPassword property of the Security.SSLConfigs class.

For more information on opening a TCP device, see [OPEN and USE Command Keywords for TCP Devices](#).

Having added TLS security to the connection, you can continue to use it in the same manner as before.

Configure a Server to Use TLS with a TCP Socket

To enable a socket to require a secure connection from a client, you can either:

- Open a TCP socket specifying that this connection requires TLS.
- Establish the requirement for the use of TLS on an already-existing socket.

Establish a TLS-secured Socket

In this scenario, InterSystems IRIS is the server and the TCP socket uses TLS from its inception. The procedure is:

1. Make sure that the configuration you wish to use is available. If it was created before InterSystems IRIS was last started, it is activated and ready for use; otherwise, you can [create a new one or edit an existing one](#).
2. Open a TCP socket that requires the use of TLS.

This socket requires the use of TLS from clients connecting to it. When a client attempts to connect to the server, the server attempts to negotiate a connection that uses TLS. If this succeeds, the connection is available for normal use and communications are secured using the negotiated algorithm. If it fails, there is no connection available for the client.

Open a TCP Socket Requiring TLS

To open a socket that requires TLS, the procedure is:

1. Specify the device that is accepting connections:

ObjectScript

```
SET MySocket = "|TCP|1000"
```

The TCP string specifies that this is a TCP device. For more information on initiating a TCP connection, see [OPEN Command for TCP Devices](#).

2. Open the connection, specifying the use of TLS with the /TLS parameter.

ObjectScript

```
OPEN MySocket:( :1000:/TLS="MyCfg" )
```

Optionally, the call can include a password for the private key file:

```
OPEN MySocket:( :1000:/TLS="MyCfg|MyPrivateKeyFilePassword" )
```

This call opens a TCP socket on port 1000 using TLS. For more information on opening a TCP device, see [OPEN and USE Command Keywords for TCP Devices](#).

Important: The ability to include a password when opening a TCP connection using TLS is for real-time interactive use only. You should *never* store a private key password persistently without protecting it. If you need to store such a password, use the PrivateKeyPassword property of the Security.SSLConfigs class.

Add TLS to an Existing Socket

This scenario assumes that a connection to the TCP socket has already been established. The procedure is:

1. Make sure that the configuration you wish to use is available. If it was created before InterSystems IRIS was last started, it is activated and ready for use; otherwise, you can [create a new one or edit an existing one](#).
2. [Use TLS to secure the existing TCP connection to the socket](#).

Secure an Existing TCP Connection to the Socket Using TLS

This involves adding TLS to an already-existing connection to a socket on a particular machine and port number. The procedure is:

1. Determine the name of the device on which the socket is open. For example, this might have been established using the following code:

```
SET MySocket = "|TCP|1000"
OPEN MySocket:( :1000)
```

The TCP string specifies that this is a TCP device. For more information on initiating a TCP connection, see [OPEN Command for TCP Devices](#).

2. Specify the use of TLS as follows with the /TLS parameter:

```
USE MySocket:( :/TLS="MyCfg")
```

where

- *MySocket* is the device previously specified
- *MyCfg* is a TLS configuration

Optionally, the call can include a password for the private key file:

```
USE MySocket:( :/TLS="MyCfg|MyPrivateKeyFilePassword")
```

For more information on opening a TCP device, see [OPEN and USE Command Keywords for TCP Devices](#).

Important: The ability to include a password when securing an existing TCP connection using TLS is for real-time interactive use only. You should *never* store a private key password persistently without protecting it. If you need to store such a password, use the PrivateKeyPassword property of the Security.SSLConfigs class.

Having added TLS security to the socket, you can continue the connection to it in the same manner as before.

TLS with the Web Gateway

Configuring the Web Gateway to Connect to InterSystems IRIS Using TLS

You can use TLS to set up a secure, encrypted channel between the Web Gateway and the InterSystems IRIS® data platform server. To do this, you need a TLS certificate and private key that represents the Gateway. The Gateway can then establish an encrypted connection to the InterSystems IRIS server (which has its own certificate and private key), so that all information is transmitted through the connection.

Note: For information on setting up a connection between the Web Gateway and the InterSystems IRIS server that is protected by Kerberos, see [Setting Up a Kerberized Connection from the Web Gateway to InterSystems IRIS](#).

The procedure is:

1. If there is not already a %SuperServer TLS configuration associated with the InterSystems IRIS server, create one as described in [Create or Edit a TLS Configuration](#).
2. On the Portal's **System-wide Security Parameters** page (**System Administration > Security > System Security > System-wide Security Parameters**), for the **Superserver SSL/TLS Support** choice, select **Enabled** or **Required**. For more details on these settings, see [System-wide Security Parameters](#).
3. Go to the Web Gateway's **Server Access** page (**System Administration > Configuration > Web Gateway Management**).
4. On that page, under **Configuration**, select **Server Access**.
5. Next, select **Edit Server** and click **Submit**. This displays the configuration page for the Web Gateway.
6. On this page, configure the Web Gateway to use TLS. Specifically, for the **Connection Security Level** field, select **SSL/TLS**. You must specify values for the **SSL/TLS Protocol** and **SSL/TLS CA Certificate File** fields. The other fields may be required or optional depending on other settings. The **SSL/TLS Certificate File** and **SSL/TLS Private Key File** are required if **Require peer certificate verification** is selected. If including a SSL/TLS private key file, you must also specify a value for the **SSL/TLS Key Type**. Additionally, if the certificate or private key file require a password, then you must provide in the **SSL/TLS Private Key Password** field either:
 - The private key password (which cannot begin with { or end with })
 - An operating system command enclosed in braces (for example, {sh /tmp/script.sh}). See [Retrieve Passwords Programmatically](#) for more details.

For more details on the fields on this page, see the [Configuring Server Access](#) section of “Web Gateway Operation and Configuration”.

Certificate Chain

Establishing the Required Certificate Chain

For a connection to be successfully established using a cipher suite that uses certificates and keys, the client must be able to verify the server's certificate chain from the server's own certificate to a self-signed certificate from a trusted certificate authority (CA), including intermediate certificates (if any). If the server is authenticating the client user, then the server must also be able to verify the client user's certificate chain from the client user's own certificate to a trusted CA's self-signed certificate, including intermediate certificates (if any).

Since authentication can be bidirectional, the requirements for certificate chains refer to the verifying entity (the side requiring the authentication) and the verified entity (the side being authenticated), rather than the client and the server.

For authentication to be possible, the following conditions must be met:

- The verifying entity must have access to all the certificates that constitute the certificate chain from the verified entity's own certificate to a trusted CA's self-signed root certificate. The certificates in the chain are obtained from the combination of the verified entity's certificate file (the certificates are sent as part of the handshake protocol) and the verifying entity's trusted CA certificate file.
- The verifying entity must have the trusted CA's self-signed root certificate in its CA certificate file.
- The verified entity's own certificate must be the first entry in its certificate file.
- All intermediate CA certificates must be present.
- The certificates in the certificate chain may be divided between the verified entity's certificate file and the verifying entity's trusted CA certificate file. However, each part must be a contiguous partial certificate chain, as described in the following example.

Suppose there are:

- A verified entity (named "VE") with a certificate signed by the certificate authority named "ICA1."
- A certificate for "ICA1" signed by the certificate authority "ICA2," and a certificate for "ICA2" signed by "RootCA".
- A trusted CA (named "RootCA") with a self-signed root certificate.

The following are valid distributions of certificates between the verified entity and the verifying entity:

Table D-1: Valid Certificate Distribution Schemes

Certificates in the Verified Entity's Certificate File	Certificates in the Verifying Entity's Trusted CA Certificate File
VE	ICA1, ICA2, RootCA
VE, ICA1	ICA2, RootCA
VE, ICA1, ICA2	RootCA

Note that it is *not* valid to have VE and ICA2 in the verified entity's certificate file and ICA1 and RootCert in the verifying entity's trusted CA certificate file

